# IoT Product Design and Rapid Prototyping

## Part B

Brian Rashap

January 2023

# Previous Capstones

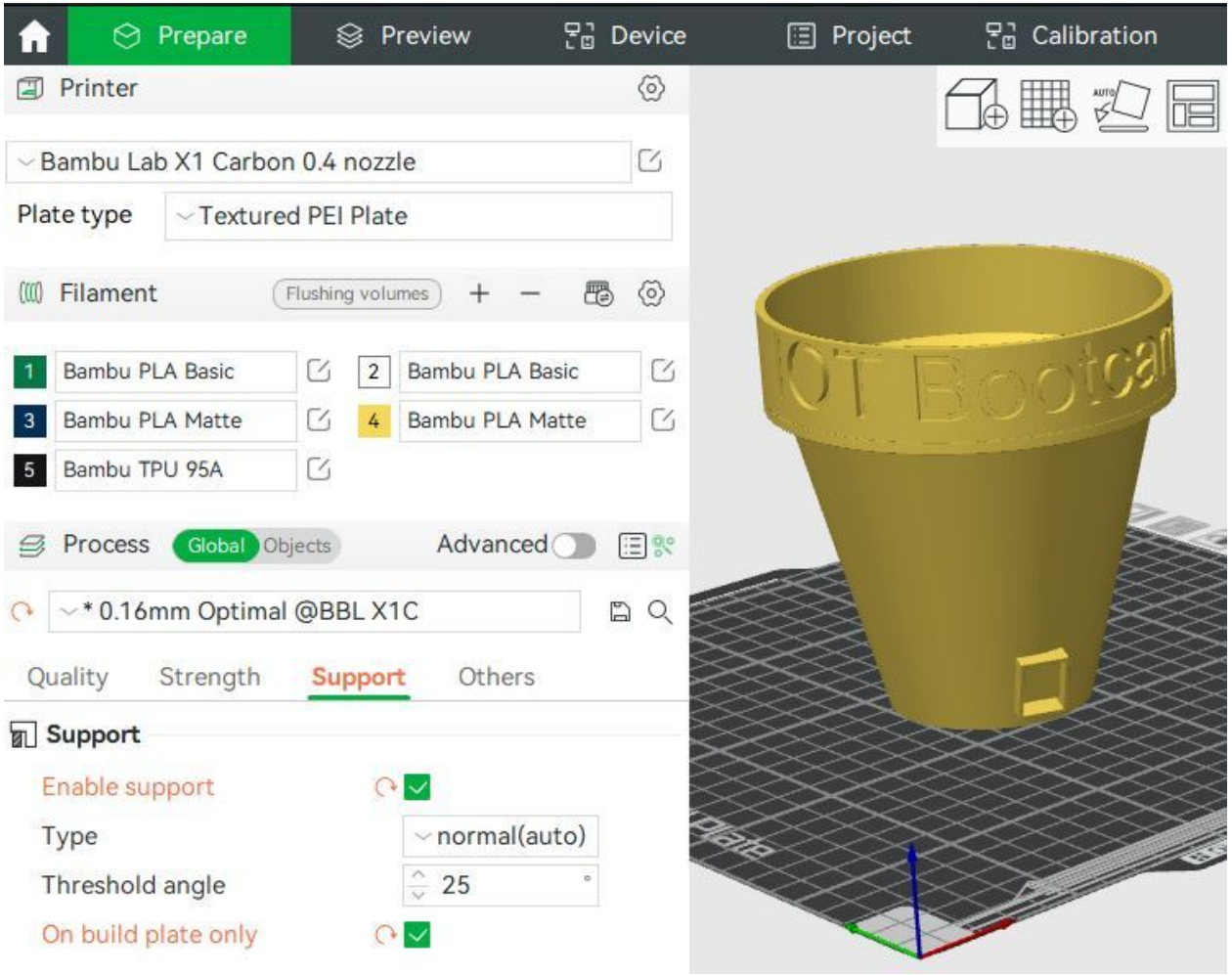

Previous Capstone Playlist

`https://www.youtube.com/watch?v=s4TslpITeVw&list=PL0t2Pk5ETDgxfVptdyr6xbL6MW1-5CJey`

# Recommendations

- Students and employers often would like references/recommendations from the instructors.
- Get permission from instructor or Deep Dive staff member before listing as a reference.
- Our recommendations are based on:
  - Ability to plan work and complete assignments on time.
  - Showing a willingness to learn and try new things.
  - Quality of documentation and use of lab notebooks
  - Willingness to help others.
  - Demonstration of learning throughout the course.

# Printing IoT Flowerpot



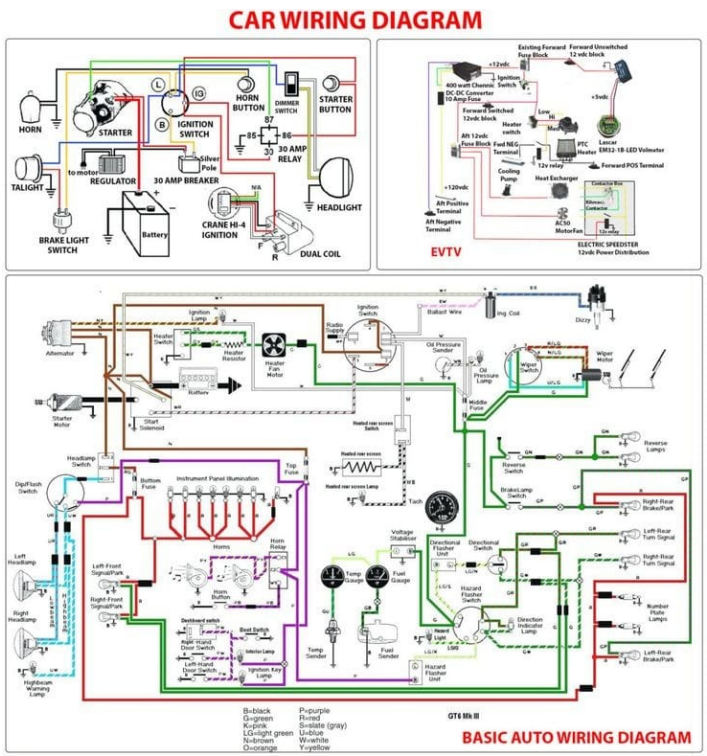

It is important to enable support as On Build plate only .

# Clear Out Credentials

Going forward we will need to connect to the Internet and the Particle Cloud:

- Run L09_00_HelloReset
  - Clear out all credentials (to remove IoTClassroom)
  - Reconnect to DDCIOT
- Add your home credentials, either by:
  - Add your home credentials to L09_00_HelloReset, or
  - When you get home, use `https://docs.particle.io/tools/developer-tools/configure-wi-fi/` to reconnect to your home wifi
- Code also will show you your stored credentials, your IP and MAC address, and the Access Point info.

# A Word About Schematics



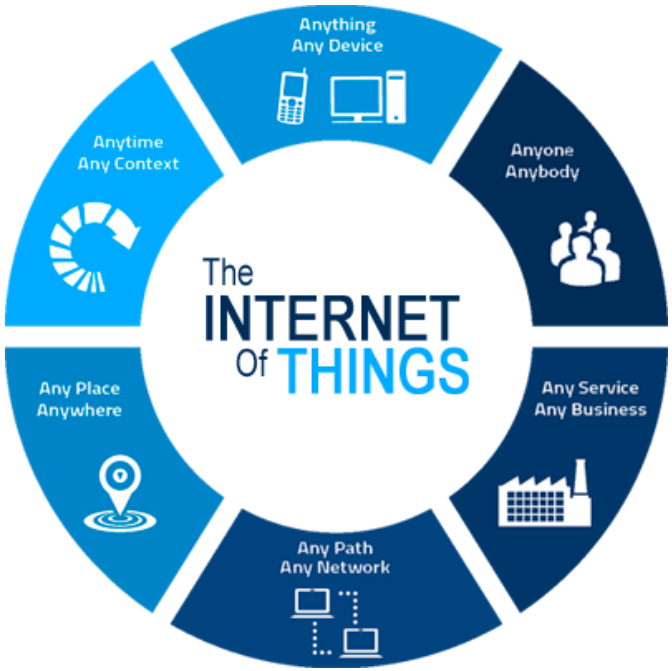

Some sensors went out on my car so I did some looking things up and ordered the parts and then downloaded the repair manual. There are pages and pages of electrical systems diagrams and thanks to you and the class and spending 10+ weeks constantly making wiring diagrams they're not terrifying and actually make sense. IoT class meets auto repair. Thank you.
– Artist from Cohort 11

# The Use of Functions: L09_00_Function


- Benefits of using functions
  1. Enables reusability and reduces redundancy
  2. Makes code modular
  3. Provides abstraction functionality
  4. Makes the program easier to understand and manage
  5. Breaks an extensive program into smaller and simpler pieces
  6. Reduces interference between variable names
- It is common during the first midterm for students to avoid the use of functions or to use global variables across functions.
- For the second midterm (Smart Houseplant Watering System) you will be asked to make appropriate use of functions.
- Review L09_00_Function. In your notebook:
  1. Document the functionality of this C++ code
  2. Note observations about how the overall code is written
  3. Identify any questions you have about the code

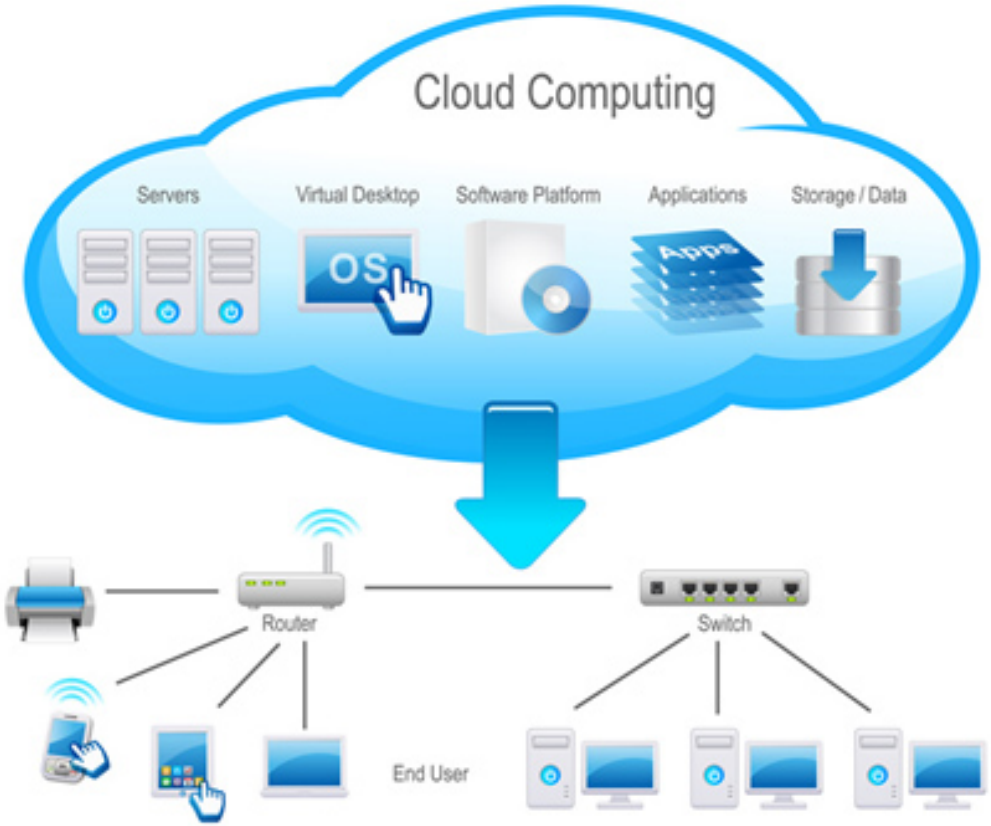
# Module 9 - The Cloud

# Module 9 Objectives

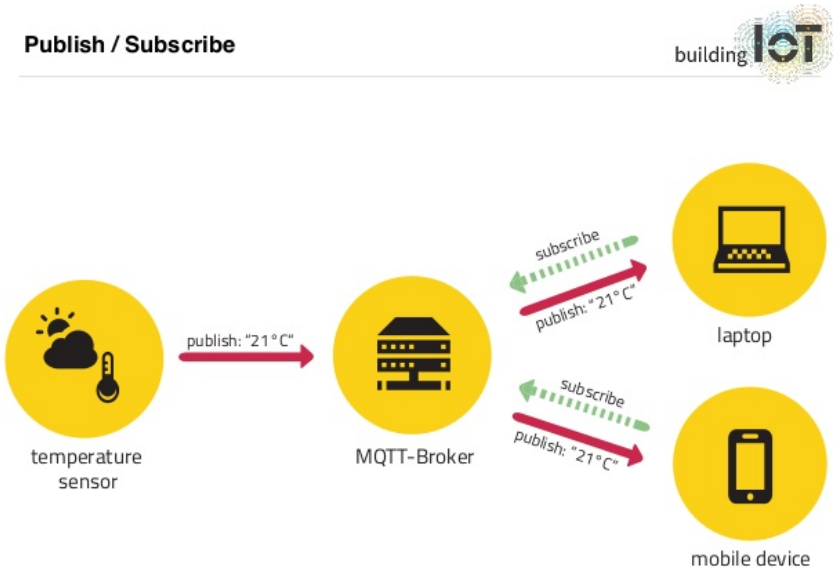


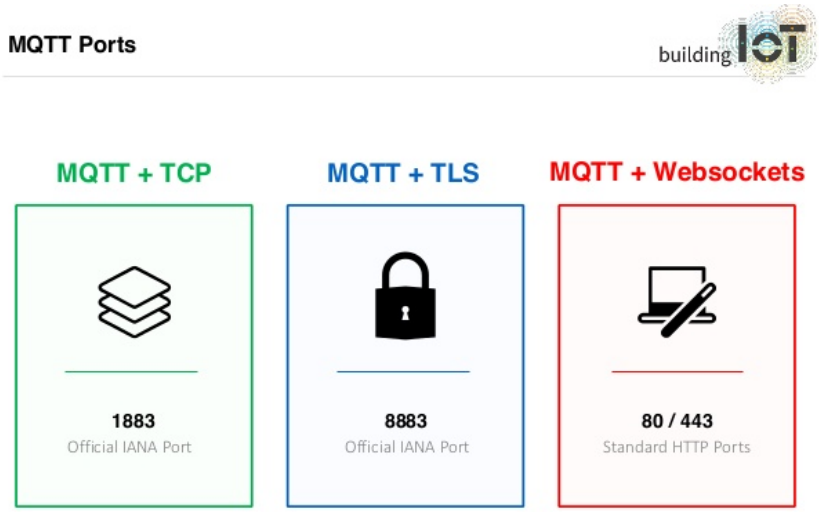

- Learning Objectives
  1. Communicating to the cloud using MQTT
  2. Javascript Object Notation (JSON)
  3. Webhooks
- Additional Items
  1. 3D Modeling Lesson 4 - Tubes
  2. Quiz 6

# The Cloud


Cloud Computing
Servers
Virtual Desktop
Software Platform
Applications
Storage / Data
OS
Apps
Router
Switch
End User

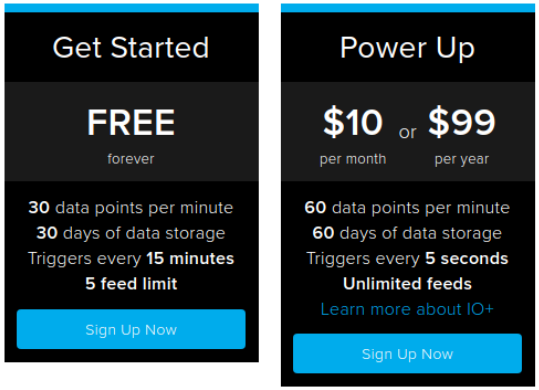
# MQTT: MQ Telemetry Transport



# Adafruit.io

Let's create an Adafruit.io account.



# Creating credentials.h

The credentials.h file allows you to store private data that will not be sent to github (if you are using the .gitignore file that was cloned with your repository).

From VSCode:

- File → Open Folder
- Select from L09_Cloud: L09_01_SubscribePublish
- Setup credentials.h:
  1. Create a new file in the /src directory called credentials.h.
  2. Type this code into it, using your own username and key:

```
1 //************************ Adafruit.io Setup ***************************************
2 #define AIO_SERVER      "io.adafruit.com"
3 #define AIO_SERVERPORT  1883         // use 1883 for SSL
4 #define AIO_USERNAME    "username"   // replace with your Adafruit.io username
5 #define AIO_KEY         "key"        // replace with your Adafruit.io key
```

# MQTT Elements explained

- TheClient - object that defines the TCP (Transmission Control Protocol) connection over WiFi.
- mqtt - object that defines the MQTT connection using the WiFi object, the MQTT server/port, and user name/password.
- FeedName - a "variable" located on Adafruit.io that can be subscribed or published to. There can be many of these.
- mqttObj - object that will be used in the C++ code that will be used to publish or subscribe to an Adafruit.io feed. There needs to be one object for each feed.
- value - Variable in the C++ code that stores information to be published or to receive information from a feed that is subscribed to.

*NOTE: FeedName, mqttObj, and value should be given descriptive "names" similar to the naming convention for all variables and objects in the C++ code.*

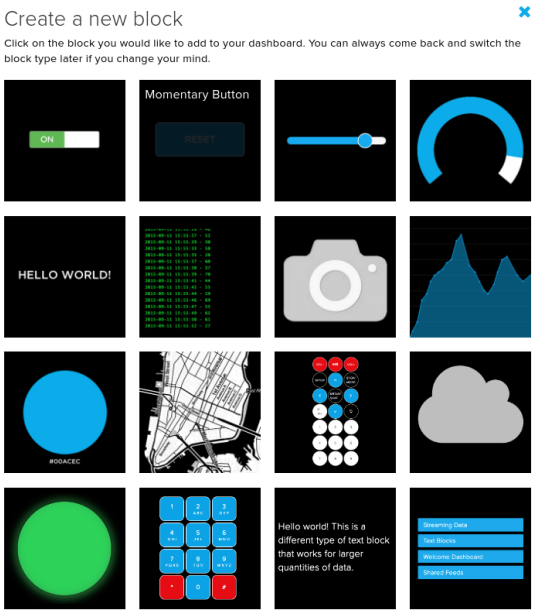
# MQTT Elements in VSCode

```
#include <Adafruit_MQTT.h>
#include "Adafruit_MQTT/Adafruit_MQTT_SPARK.h"
#include "Adafruit_MQTT/Adafruit_MQTT.h"

#include "credentials.h"

// Create the TCP Client
TCPClient TheClient;

// Setup the MQTT client class by passing in the WiFi client and MQTT server and login details.
Adafruit_MQTT_SPARK mqtt(&TheClient,AIO_SERVER,AIO_SERVERPORT,AIO_USERNAME,AIO_KEY);

/****************************** Feeds ***************************************/
// Setup Feeds to publish or subscribe
// Notice MQTT paths for AIO follow the form: <username>/feeds/<feedname>
Adafruit_MQTT_Subscribe subFeed = Adafruit_MQTT_Subscribe(&mqtt,AIO_USERNAME "/feeds/feed1");
Adafruit_MQTT_Publish pubFeed = Adafruit_MQTT_Publish(&mqtt,AIO_USERNAME "/feeds/feed2");

/************Declare Variables*************/
unsigned int last, lastTime;
float subValue,pubValue;

/************Declare Functions*************/
void MQTT_connect();
bool MQTT_ping();
```

# MQTT Publish and Subscribe

```
void setup() {
  Serial.begin(9600);
  waitFor(Serial.isConnected, 15000); //wait for Serial Monitor to startup

  WiFi.connect();                      //Connect to internet, but not Particle Cloud
  while(WiFi.connecting()) {
    Serial.printf(".");
  }

  mqtt.subscribe(&subFeed);           // Setup MQTT subscription for subFeed feed.
}

void loop() {
  // Publishing to a MQTT feed
  if(mqtt.Update()) {   //if mqtt object (Adafruit.io) is available to receive data
    Serial.printf("Publishing %0.2f to Adafruit.io feed FeedNameB \n",value1);
    pubfeed.publish(value1);
  }
    // Two new functions that will be useful:
    // atof() - ASCII to Float: converts an ASCII string to a floating point number
    // atoi() - ASCII to Integer: converts an ASCII string to an integer

  // Receive data from a subscription to an MQTT feed
  Adafruit_MQTT_Subscribe *subscription;
  while ((subscription = mqtt.readSubscription(100))) { //wait a moment for new feed data
    if (subscription == &subFeed) {  // assign new data to appropriate variable
      value2 = atof((char *)subFeed.lastread); //value2 = data from MQTT subscription
      Serial.printf("Received %0.2f from Adafruit.io feed FeedNameB \n",value2);
    }
  }
}
```

# Assignment: L09_01_SubscribePublish



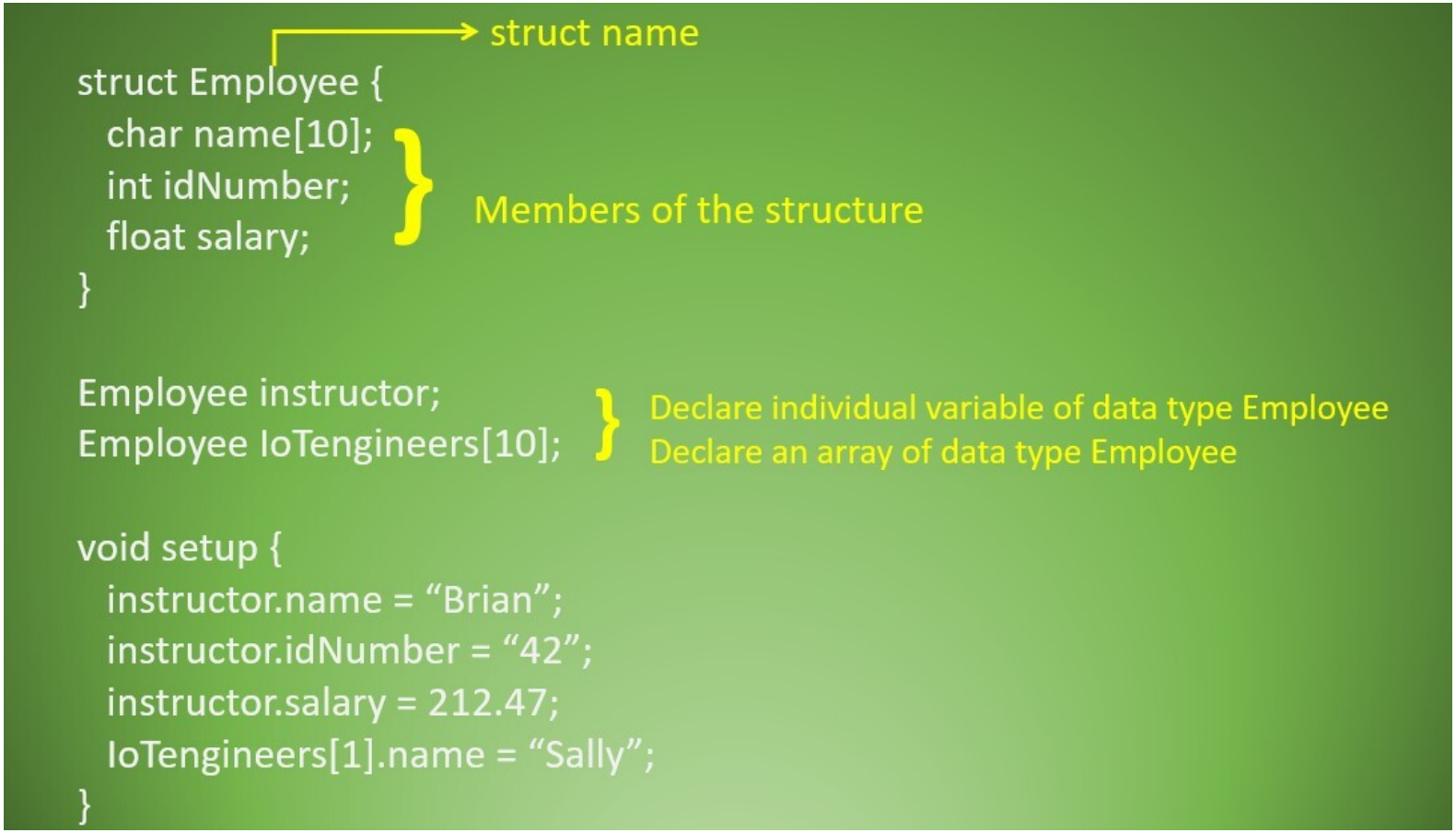

1. Modify the starter code for your Adafruit.io
2. Publish
   - Publish a random number to a feed once every 6 seconds (do not use a delay).
   - Create a line chart on your dashboard to display the random number.
3. Subscribe
   - Add a button to your dashboard and connect it to a feed called buttonOnOff.
   - Subscribe to the buttonOnOff and turn on the on board LED (D7) when pressed.
4. Experiment with other blocks
   - Add a LED to pin D16.
   - Add a slider and new feed.
   - Control the brightness of an LED.
   - Display data with other dashboard blocks.

# Local and Static Variables

```
1  const int TEMPFREQ = 10000, MOISTFREQ = 30000, MOISTPIN = A3;
2  float tempC;
3  int moist;
4  void loop() {
5    tempC = getTemp(TEMPFREQ);
6    moist = getMoisture(MOISTPIN,MOISTFREQ);
7  }
8
9  float getTemp(int timeInterval) {
10   int currentTime;
11   static int lastTime = -999999;
12   static float data;
13
14   currentTime = millis();
15   if(currentTime - lastTime > timeInterval) {
16     lastTime = millis();
17     data = BME.readTemperature();
18   }
19   return data;
20 }
21
22 int getMoisture(int probePIN, int timeInterval) {
23   static int lastTime = -999999;
24   static int data;
25
26   if(millis() - lastTime > timeInterval) {
27     lastTime = millis();
28     data = analogRead(probePIN);
29   }
30   return data;
31 }
```

# STRUCT datatype and operators

struct enables the programmer to create a variable that structures a selected set of data.



# STRUCT datatype and operators

struct creates a variable that structures a set of data.

```
1  struct GeoLocation {    // create a struct of name geo that hold GPS data
2    float lat;
3    float lon;
4    int alt;
5  }; // ends with a ; as struct can also declare variables while being declared
6
7  GeoLocation myLoc;      //declare a variable called myLoc of type geo
8  GeoLocation locations[13];  //declare an array of type geo
9
10 void setup() {
11   //initialize myLoc with latitude, loggitude, and altitude
12   myLoc.lat = 35.120606;
13   myLoc.lon = -106.65818;
14   myLoc.alt = 1517;
15
16   Serial.printf("Location: lat %f, lon %f, alt %i \n",myLoc.lat,myLoc.lon,myLoc.alt);
17 }
```

The . (dot) operator and the $->$ (arrow) operator are used to reference individual members of structures.

- The dot operator is applied to the actual object.
- The arrow operator is used with a pointer to the object (we will learn about Pointers in Lesson 12).

# JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.
JSON is built on two structures:

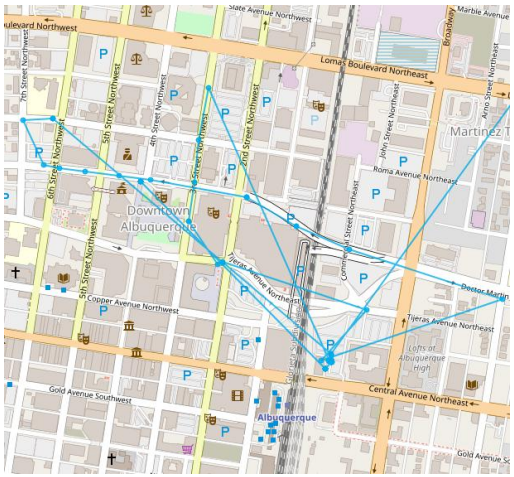
- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

# JSON Parser Generator

Creating objects in JSON are straightforward but can be tedious. There is a JSON Generator available to simplify the process.

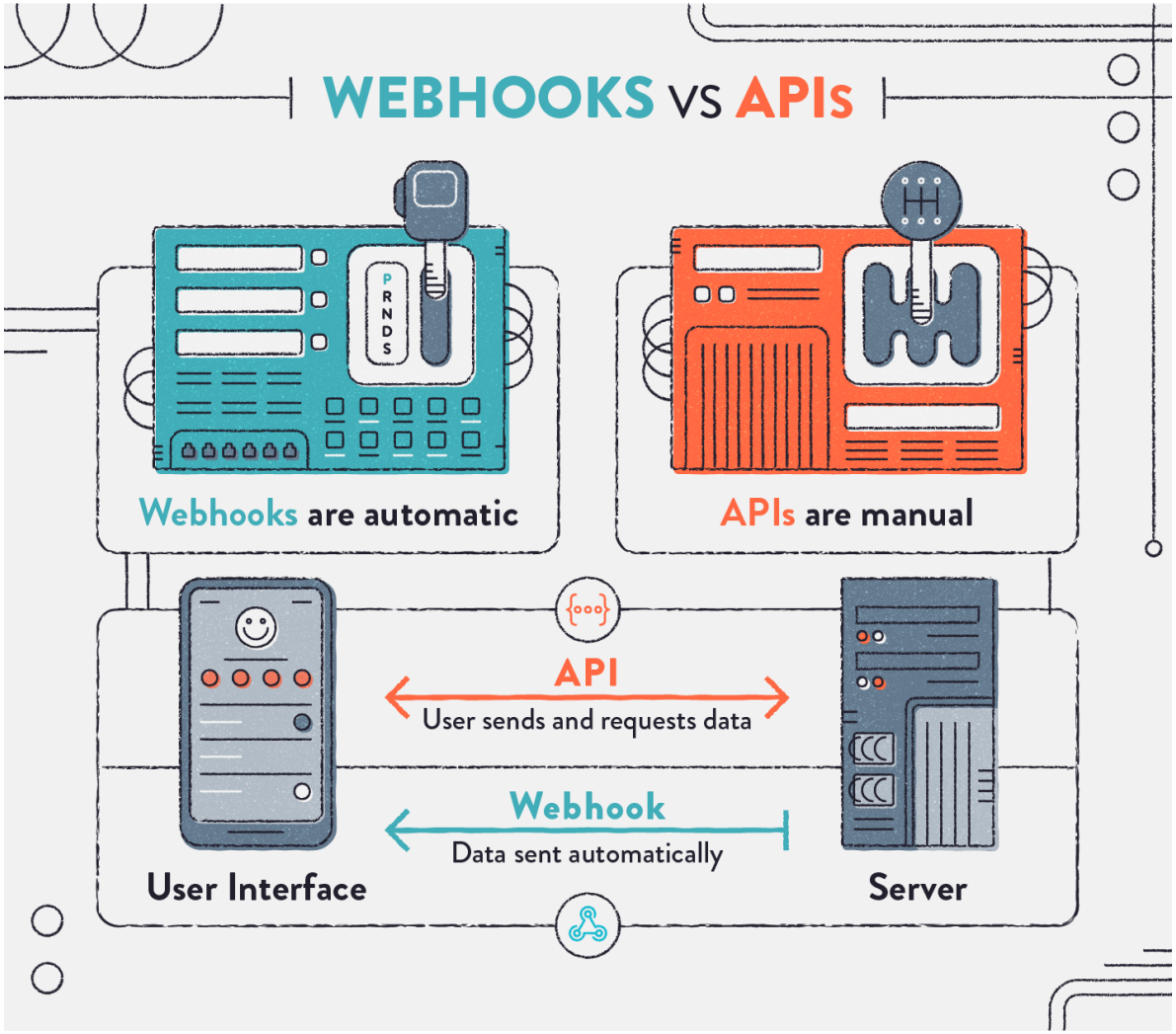
```
#include "JsonParserGeneratorRK.h"

void createEventPayLoad(float tempValue, float presValue, float humValue) {
  JsonWriterStatic<256> jw;
  {
    JsonWriterAutoObject obj(&jw);

    jw.insertKeyValue("Temperature", tempValue);
    jw.insertKeyValue("Pressure", presValue);
    jw.insertKeyValue("Humidity", humValue);
  }
  Particle.publish("env-vals",jw.getBuffer(), PRIVATE);
}
```

# Assignment: L09_02_GPSPublish

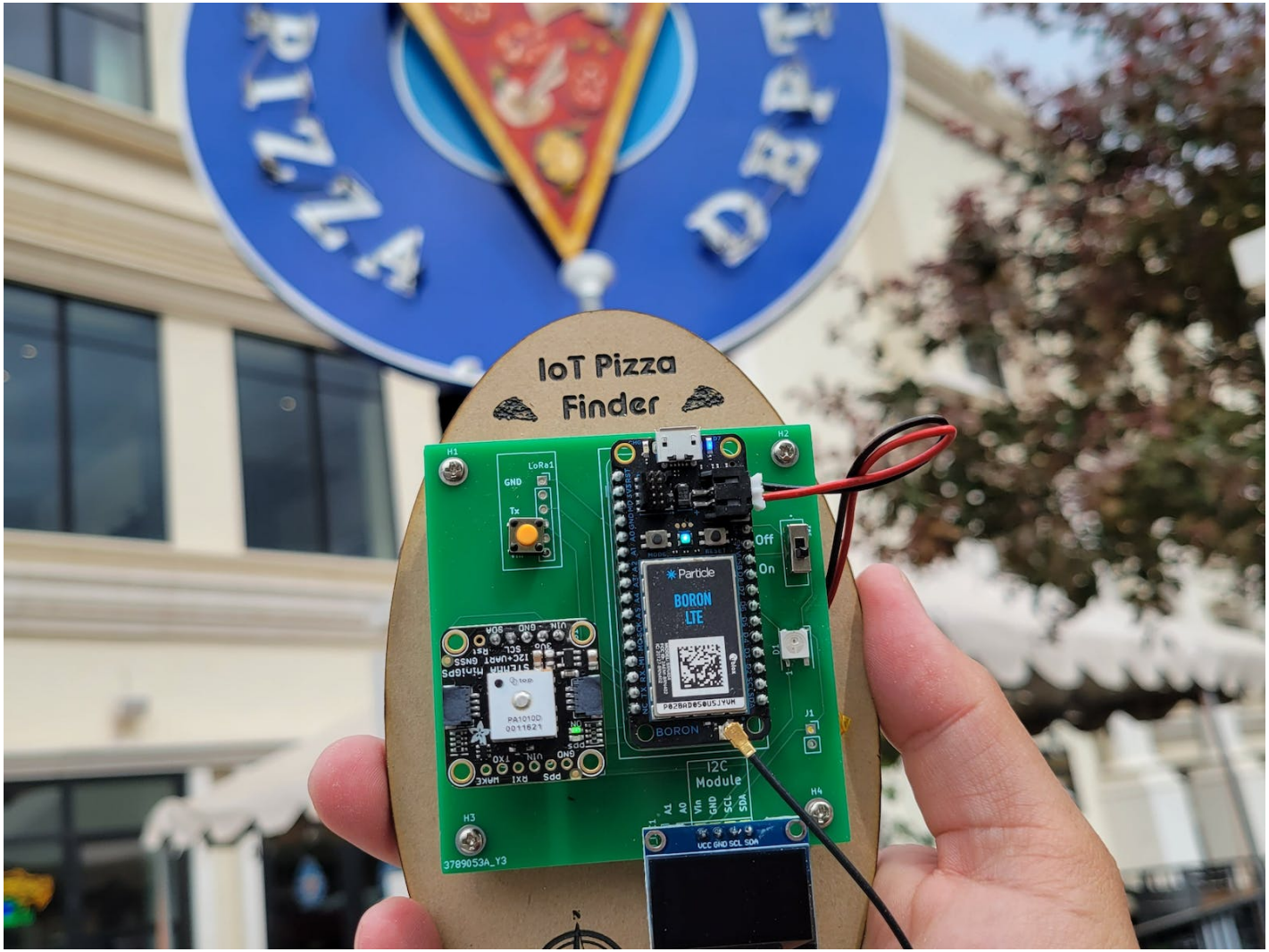



1. Create a STRUCT to hold a GPS location (latitude and longitude)
2. Every 10 seconds (without using delays), generate random GPS coordinates within Albuquerque.
3. Create a function that has as a single parameter a GPS location STRUCT variable. The function should:
   - Create a JSON playload from the GPS coordinates members of the STRUCT.
   - Publish to Adafruit.io using MQTT and JSON format. Must use "lat" and "lon" as the names for the JSON data elements.
4. Using the Map block to create a dashboard that shows the GPS coordinates

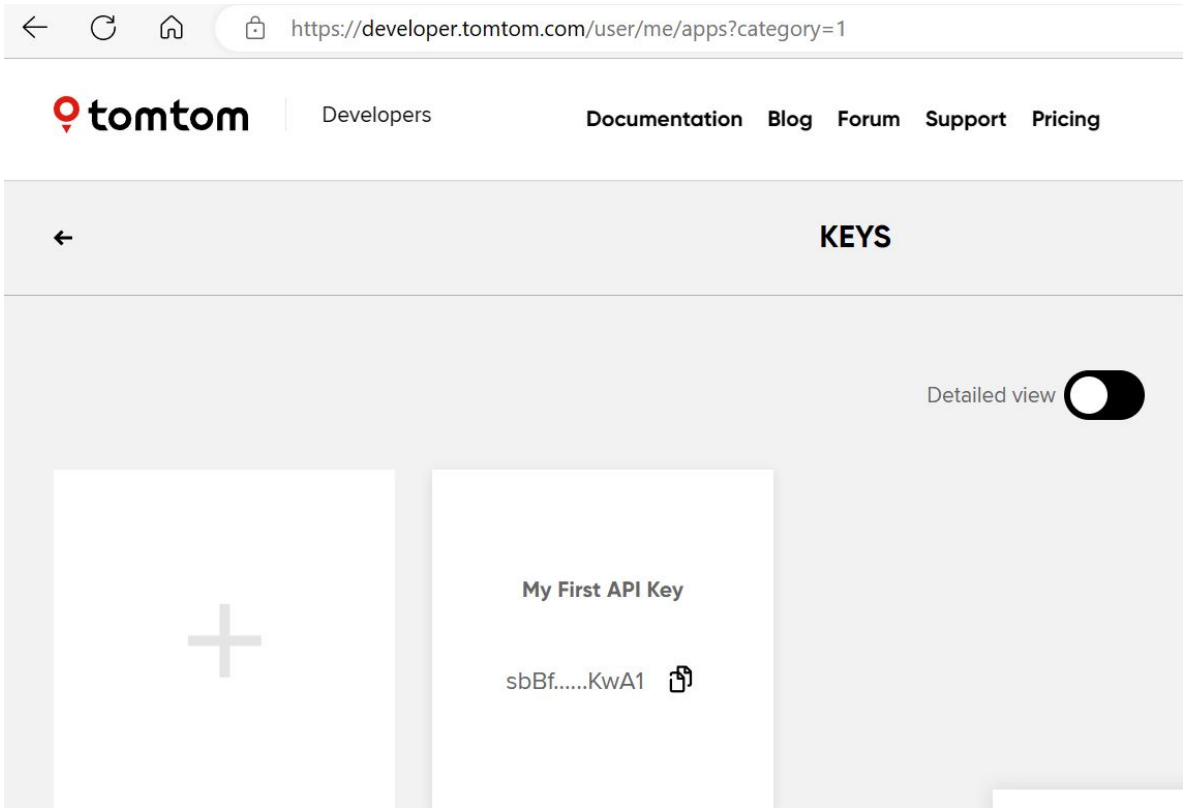
# Webhooks


WEBHOOKS vs APIs
P
R
N
D
S
Webhooks are automatic
APIs are manual
API
User sends and requests data
Webhook
Data sent automatically
User Interface
Server

# Pizza Finder



# TomTom API




Obtain a TomTom API key from
`https://developer.tomtom.com/user/register`

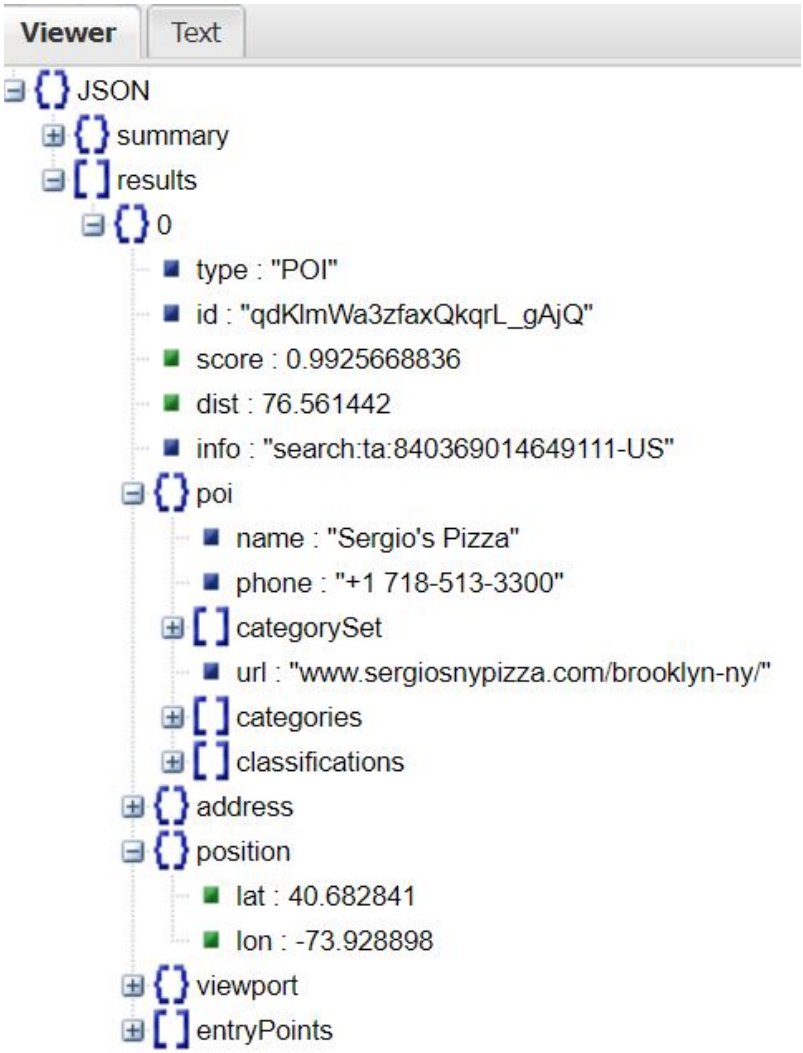
# TomTom API Call

https://api.tomtom.com/search/2/search/pizza.json?key=YOURKEY&lat=40.68353&lon=-73.92890&limit=1


```
{"summary":{"query":"pizza","queryType":"NON_NEAR","queryTime":115,"numResults":1,"offset":0,"totalResults":99,"fuzzyLevel":1,"geoBias":{"lat":40.68353,"lon":-73.9289},"queryIntent":[],"geobiasCountry":"US"},"results":[{"type":"POI","id":"qdKlmWa3zfaxQkqrL_gAjQ","score":0.9925668836,"dist":76.561442,"info":"search:ta:840369014649111-US","poi":{"name":"Sergio's Pizza","phone":"+1 718-513-3300","categorySet":[{"id":7315036}],"url":"www.sergiosnypizza.com/brooklyn-ny/","categories":["pizza","restaurant"],"classifications":[{"code":"RESTAURANT","names":[{"nameLocale":"en-US","name":"restaurant"},{"nameLocale":"en-US","name":"pizza"}]}]},"address":{"streetName":"Malcolm X Boulevard","municipalitySubdivision":"Brooklyn","municipality":"New York","countrySecondarySubdivision":"Kings","countrySubdivision":"NY","countrySubdivisionName":"New York","postalCode":"11233","extendedPostalCode":"11233-1117","countryCode":"US","country":"United States","countryCodeISO3":"USA","freeformAddress":"Malcolm X Boulevard, Brooklyn, NY 11233","localName":"Brooklyn"},"position":{"lat":40.682841,"lon":-73.928898},"viewport":{"topLeftPoint":{"lat":40.68374,"lon":-73.93008},"btmRightPoint":{"lat":40.68194,"lon":-73.92771}},"entryPoints":[{"type":"main","position":{"lat":40.68281,"lon":-73.92904}}]}]}
```

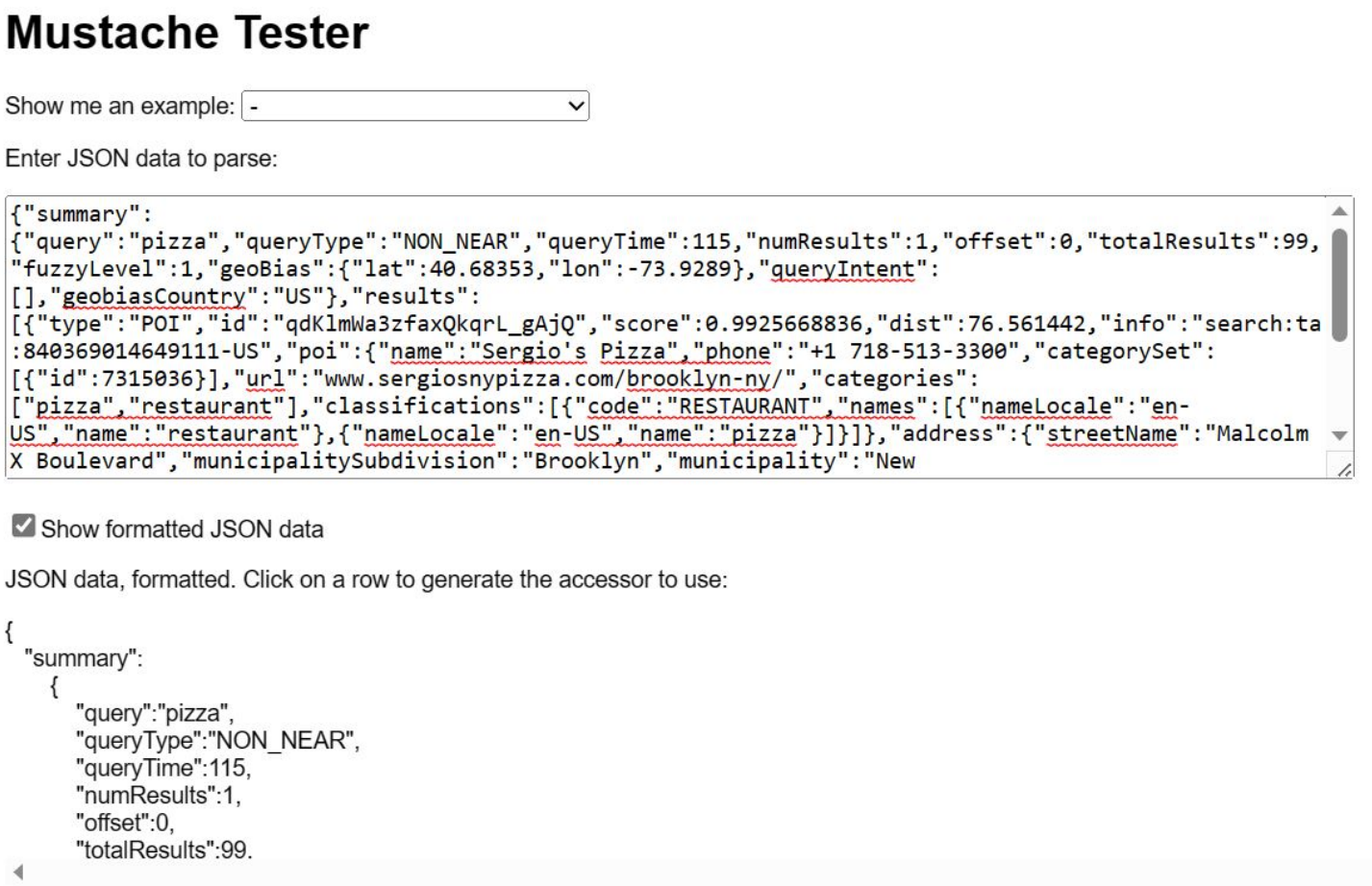
# JSON Viewer

https://jsonviewer.stack.hu/



# JSON Response Template - Mustache Tester

http://rickkas7.github.io/mustache/



# Mustache Template Test

```
1 {"name":"{{{results.0.poi.name}}}","plat":{{results.0.
    position.lat}},"plon":{{results.0.position.lon}} }
```

Enter a mustache template to test:

```
{"name":"{{{results.0.poi.name}}}","plat":{{results.0.position.lat}},"plon":{{results.0.position.lon}}
}
```

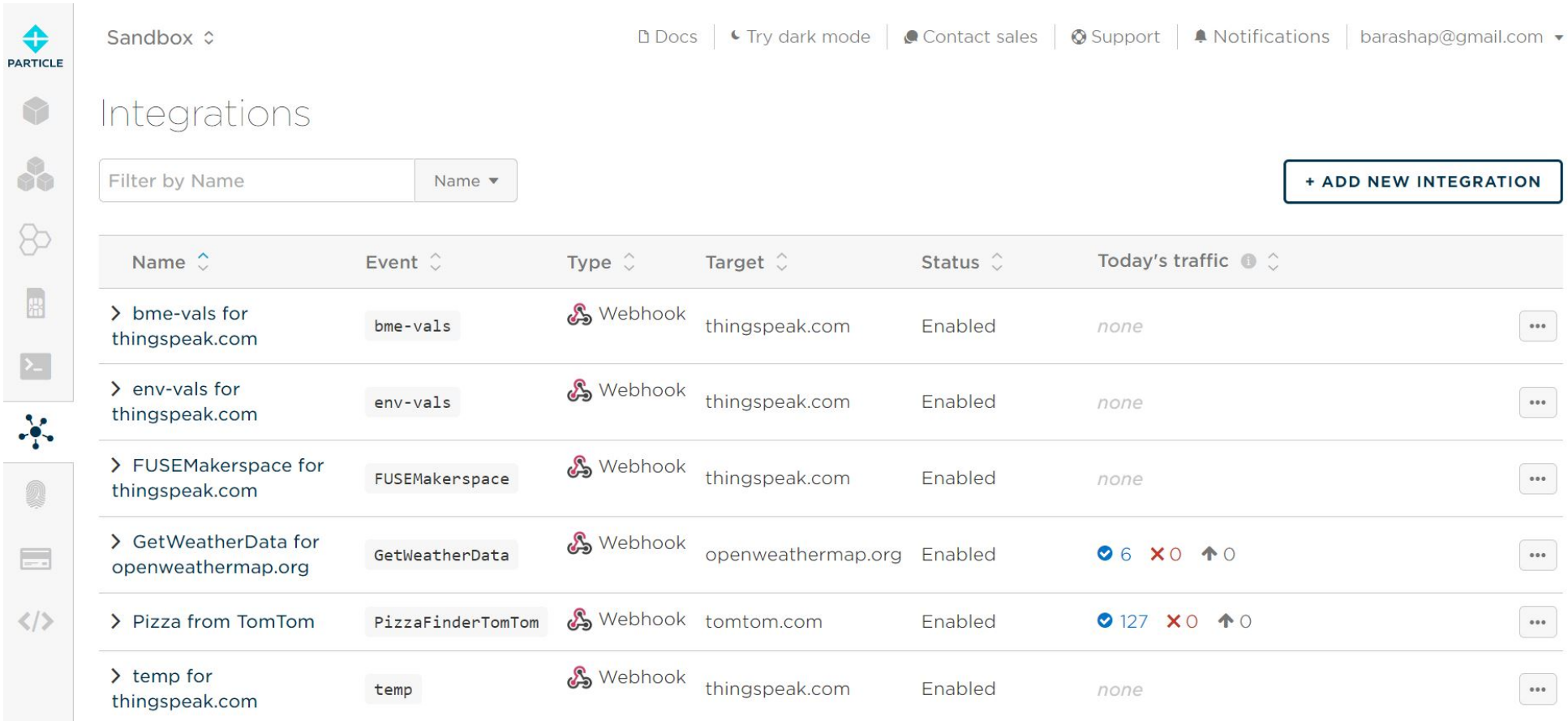
Processed template and data:

```
{"name":"Sergio's Pizza","plat":40.682841,"plon":-73.928898 }
```

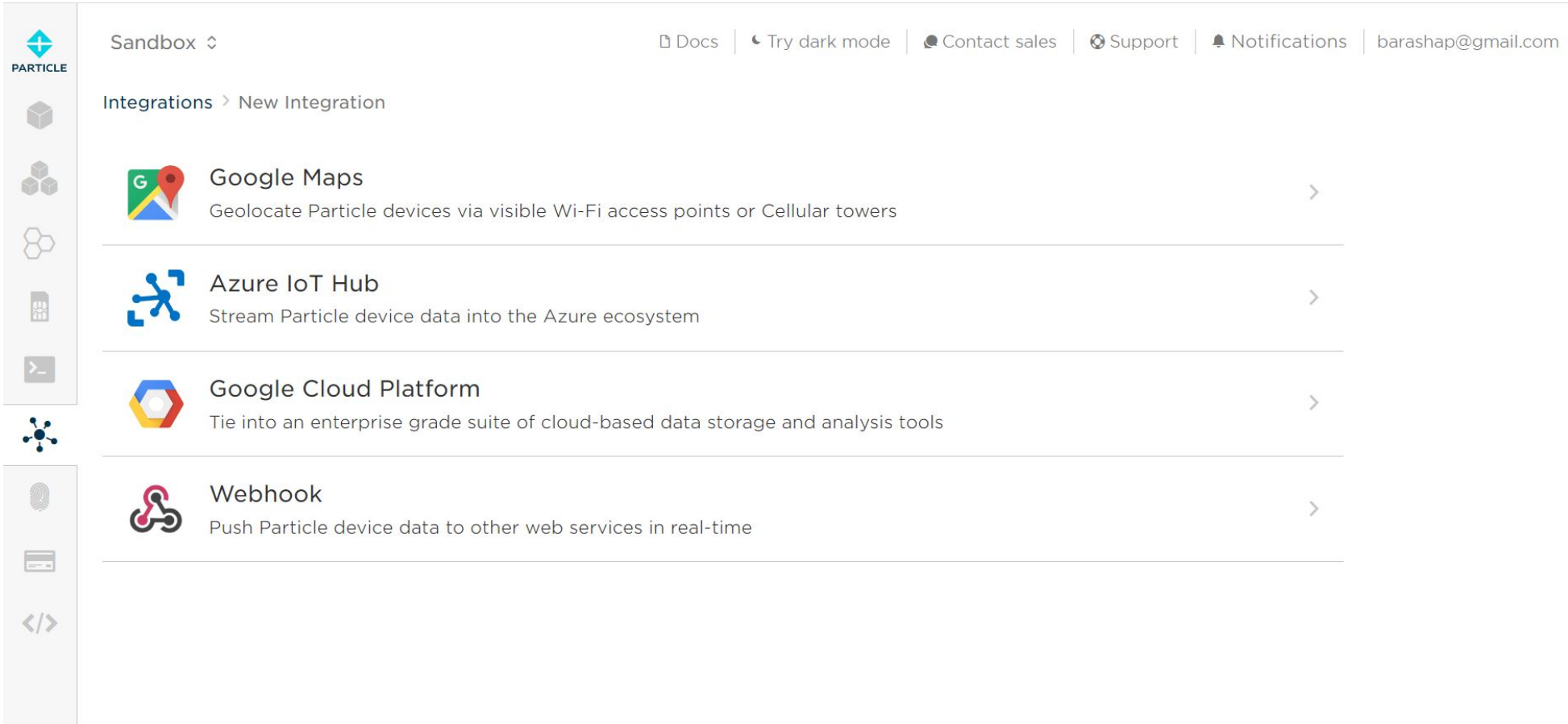
Processed template and data, as formatted JSON:

```
{
    "name": "Sergio's Pizza",
    "plat": 40.682841,
    "plon": -73.928898
}
```

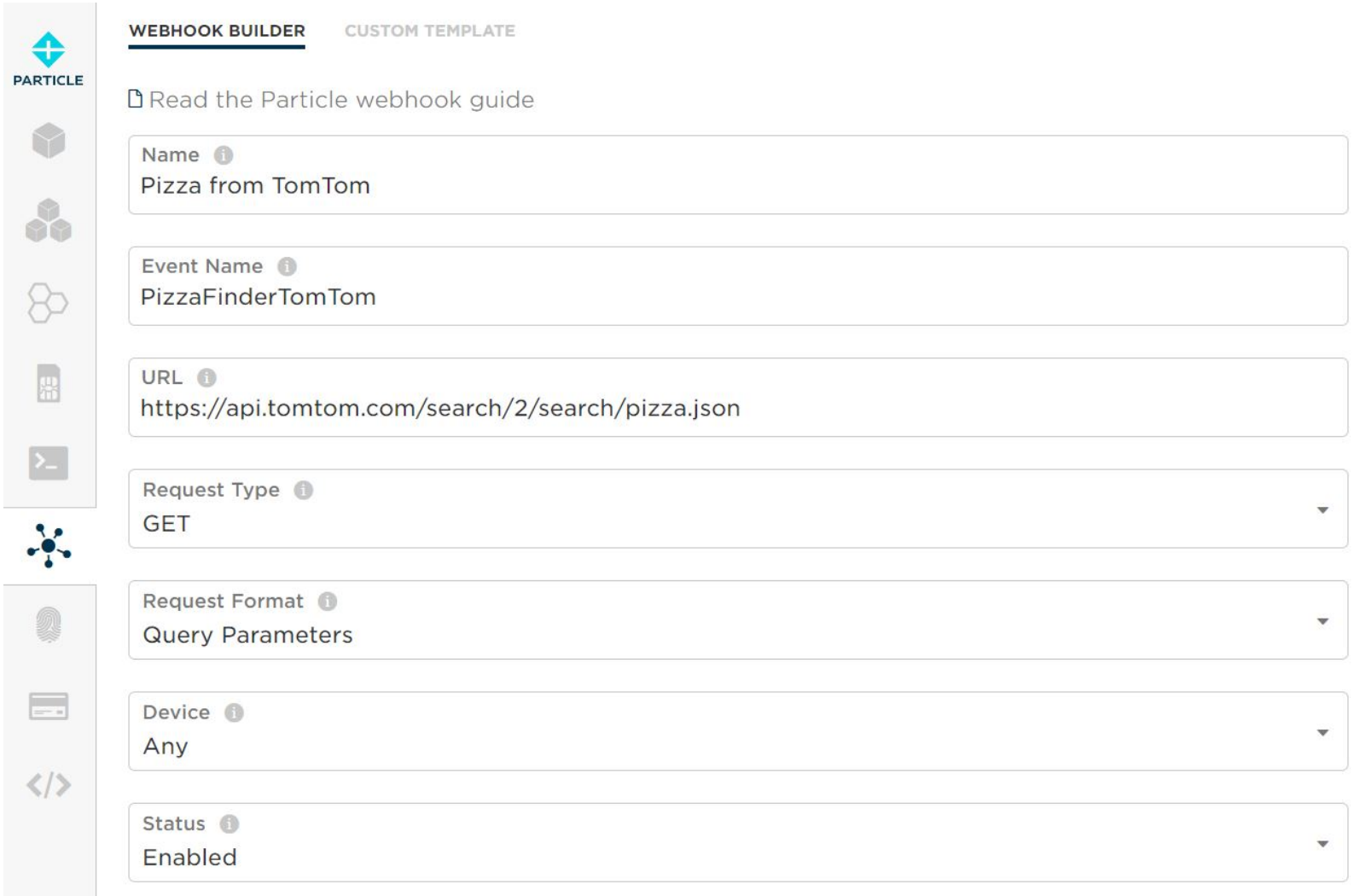
# Particle Console Integrations



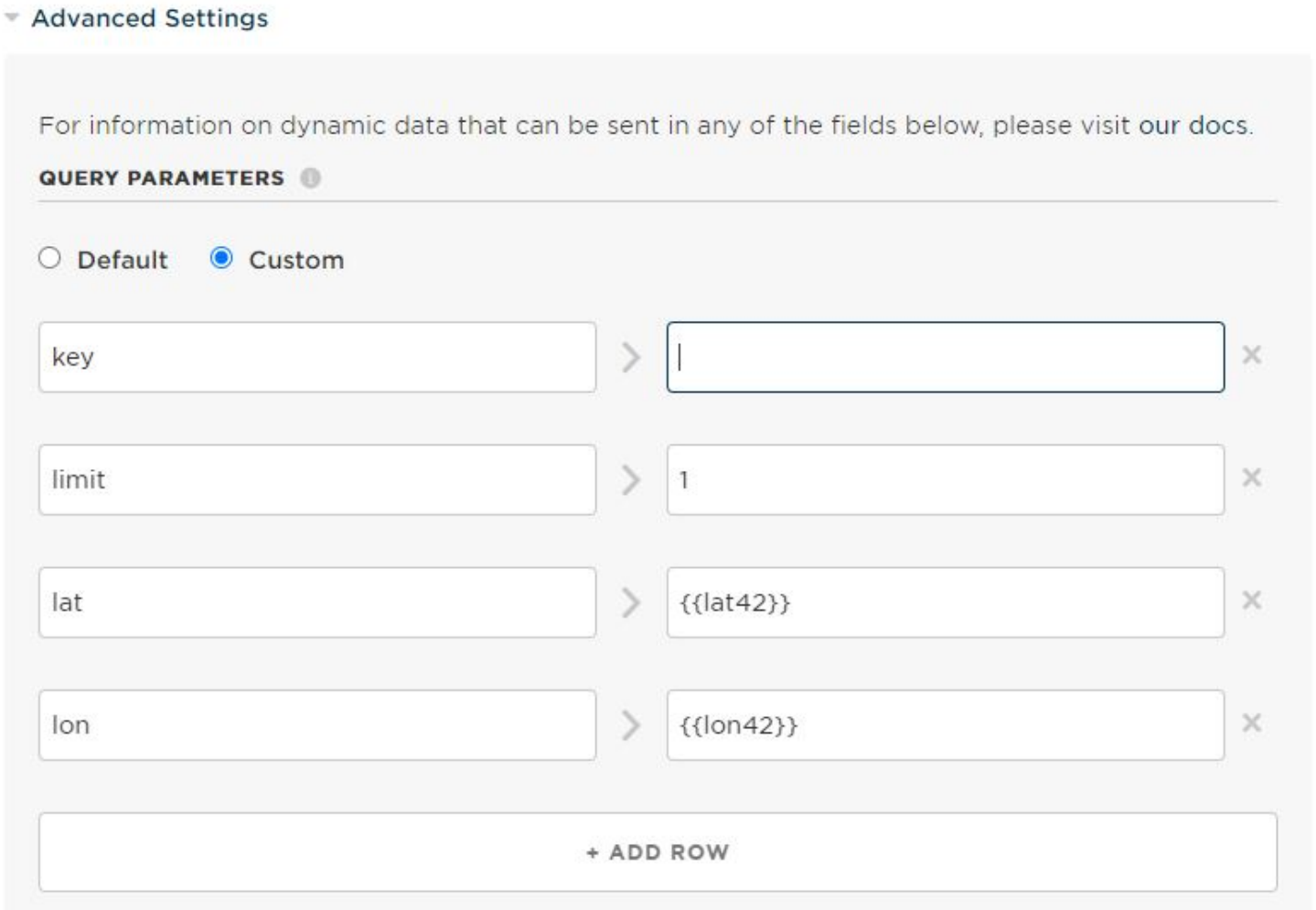
# Create a new webhook



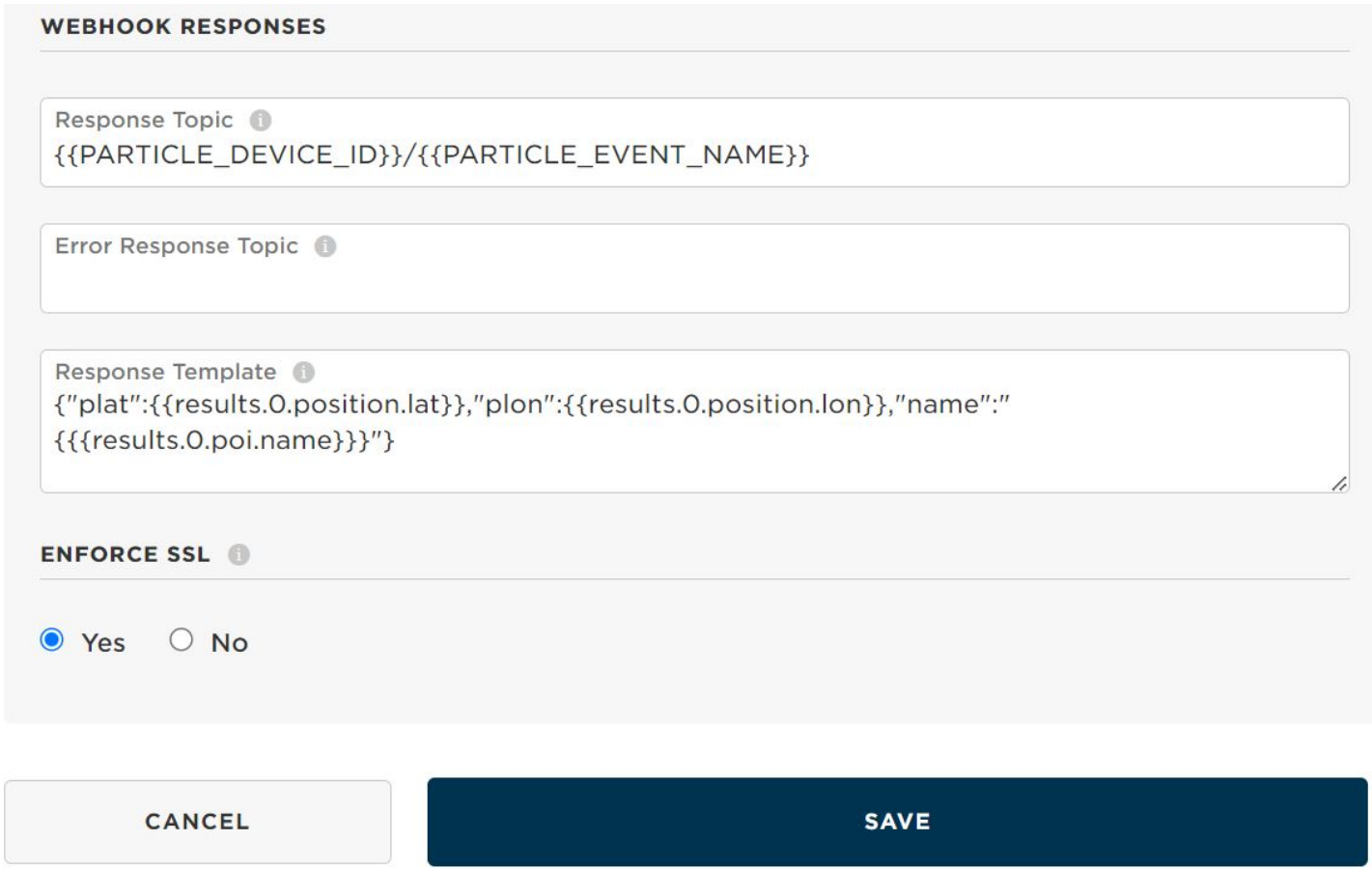
# Create a new webhook



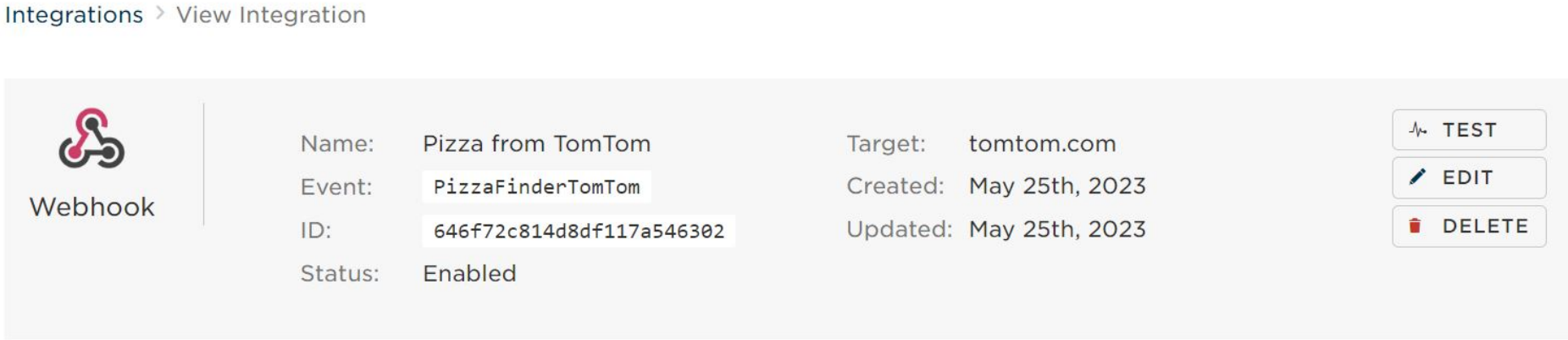
# Create a new webhook



# Create a new webhook



# Test your webhook



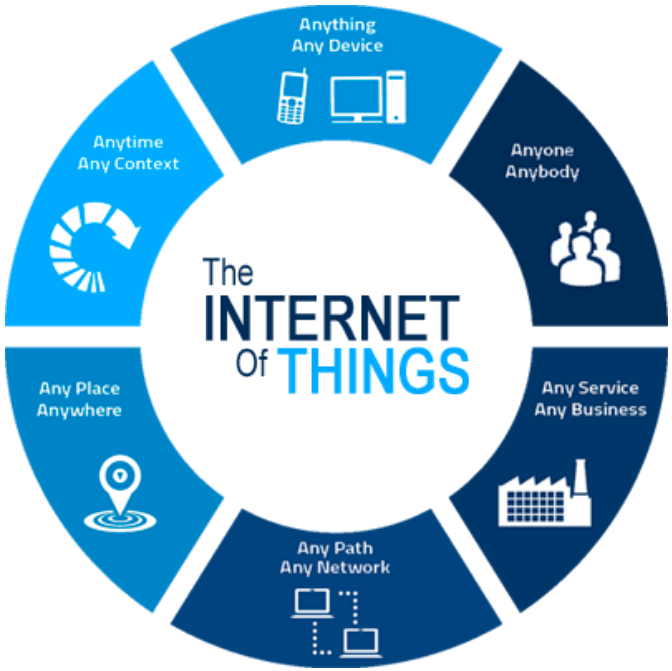
# Webhook Call and Subscription Handler

```
#include "Particle.h"
const char *EVENT_NAME = "PizzaFinderTomTom";
const float lat=40.69017, lon=-73.96359;
unsigned int lastTime;
void subscriptionHandler(const char *event, const char *data);
SYSTEM_MODE(AUTOMATIC); //must be in automatic for Particle publish/subscribe
void setup() {
  String subscriptionName=String::format("%s/%s/",System.deviceID().c_str(),EVENT_NAME);
  Particle.subscribe(subscriptionName,subscriptionHandler,MY_DEVICES);
  Serial.printf("Subscribing to %s\n",subscriptionName.c_str());
}
void loop() {
    if((millis() - lastTime) > 60000) {
        Particle.publish(EVENT_NAME, String::format("{\"lat42\":%0.5f,\"lon42\":%0.5f}",
     lat, lon), PRIVATE);
        lastTime = millis();
    }
}
void subscriptionHandler(const char *event, const char *data) {
    String pizzaName;
    float lat,lon;
    JSONValue outerObj = JSONValue::parseCopy(data);
    JSONObjectIterator iter(outerObj);
    while(iter.next()) {
        if (iter.name() == "plat") lat = iter.value().toDouble();
        if (iter.name() == "plon") lon = iter.value().toDouble();
        if (iter.name() == "name") pizzaName = (const char *)iter.value().toString();
    }
    Serial.printf("Pizza found at %s (%0.6f,%0.6f)\n",pizzaName.c_str(),lat,lon);
}
```

# Assignment: L09_03_WebHook

1. Register and obtain your TomTom API key
2. Call the TomTom API in a browser and view results in JSON Viewer and Mustache Template Builder
3. Create a webhook to find the closest pizza (or business of your choice)
4. Set the GPS coordinates to your favorite vacation location
5. Write Particle code to call your webhook and print the returned values to the screen.

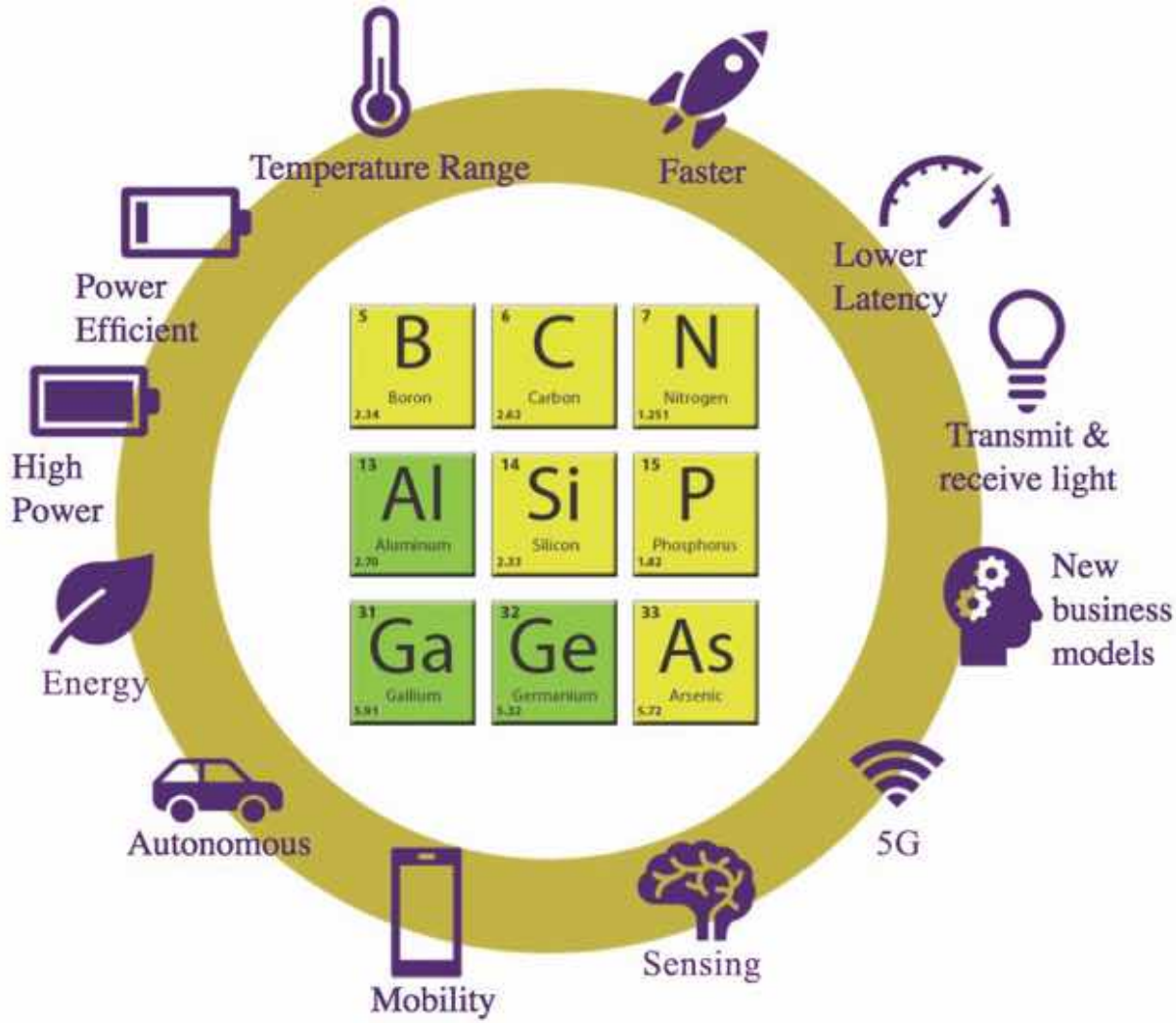
# Module 9 Review



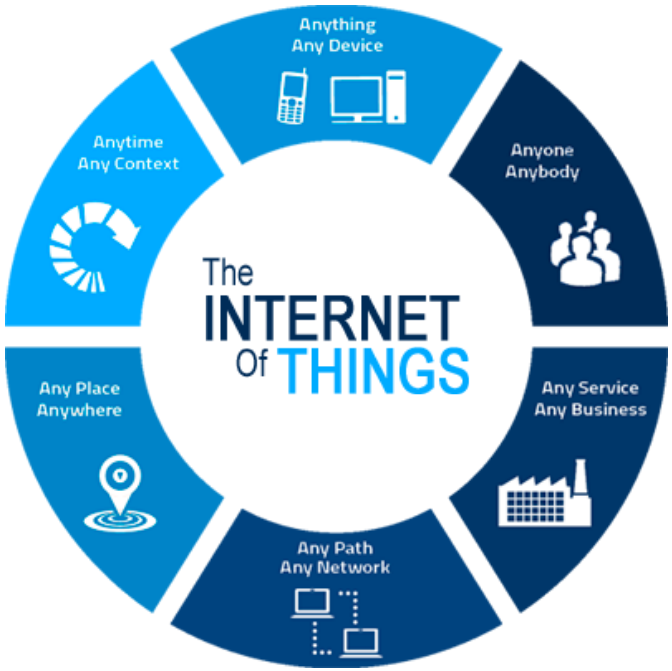

- Learning Objectives
  1. Communicating to the cloud using MQTT
  2. Javascript Object Notation (JSON)
  3. Webhooks
- Additional Items
  1. 3D Modeling Lesson 4 - Tubes
  2. Quiz 6

# Module 10 - Semiconductors

# Semiconductors



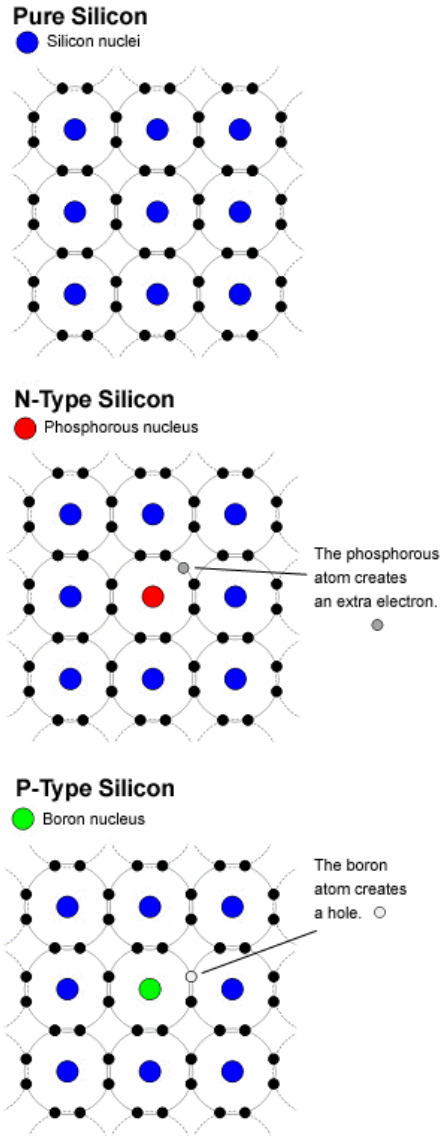
# Module 10 Objectives




- Learning Objectives
  1. Semiconductors
  2. Diodes
  3. Transistors
  4. Amplifiers

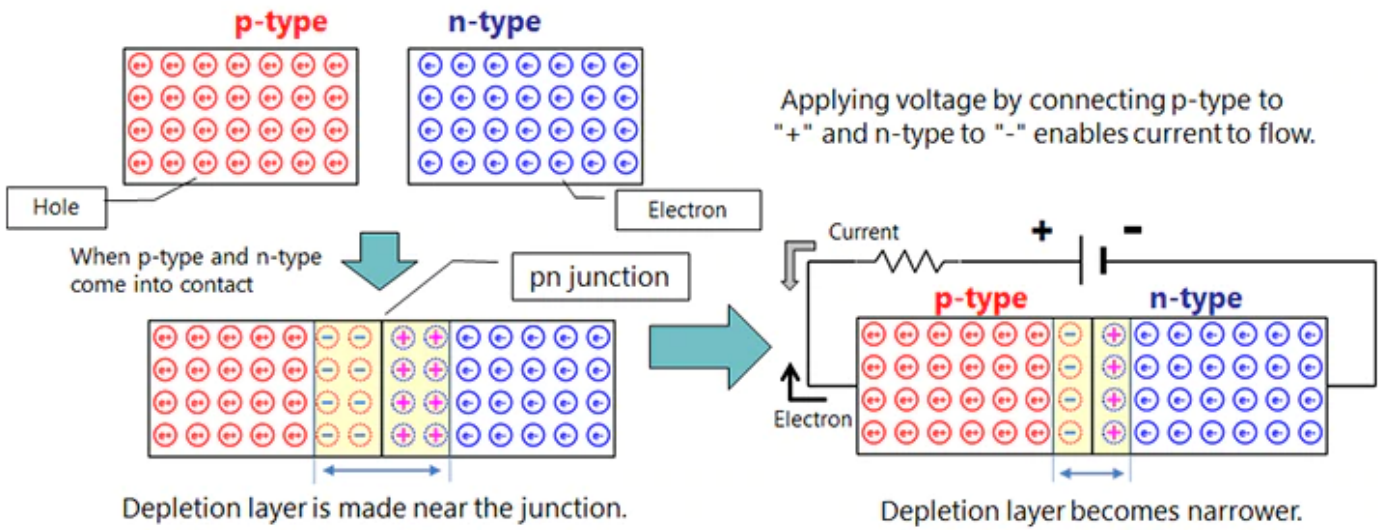
# Semiconductor

- A silicon atom has four electrons in its outer shell and bonds tightly with four surrounding silicon atoms creating a crystal matrix with eight electrons in the outer shells. The tight bonds make pure silicon non-conducting.
- Phosphorus has five electrons, and when combined, the fifth electron becomes a "free" electron that moves easily within the crystal when a voltage is applied.
- Boron has only three electrons in its outer shell and can bond with only three of surrounding silicon atoms. Thus one silicon atom has a vacant location in its outer shell, called a "hole," that readily accepts an electron.



# pn junction diode

- When p-type and n-type semiconductors are bonded, holes and free electrons are attracted, combine, and disappear near the boundary. Since there are no carriers in this area, it is called a depletion layer and it is an insulator.
- A positive voltage applied to the p-type region causes electrons to flow sequentially from the n-type. The electrons will first disappear by combining with holes, but excess electrons will move to the positive pole and current will flow.

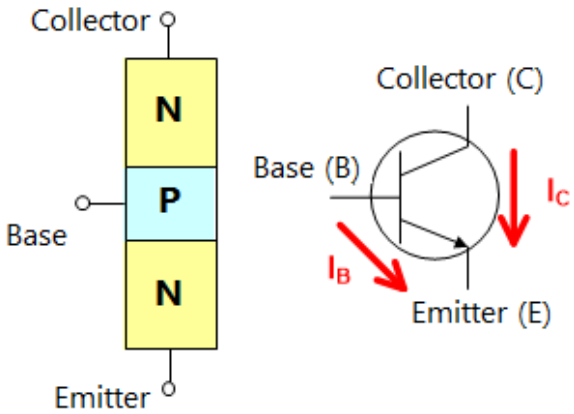


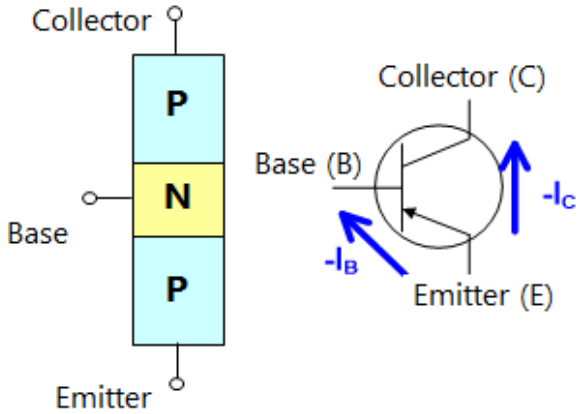
# The story of the Blue LED


https://youtu.be/AF8d72mA41M?si=dGNIvEEUrIV7rAS0

# Bipolar Junction Transistor

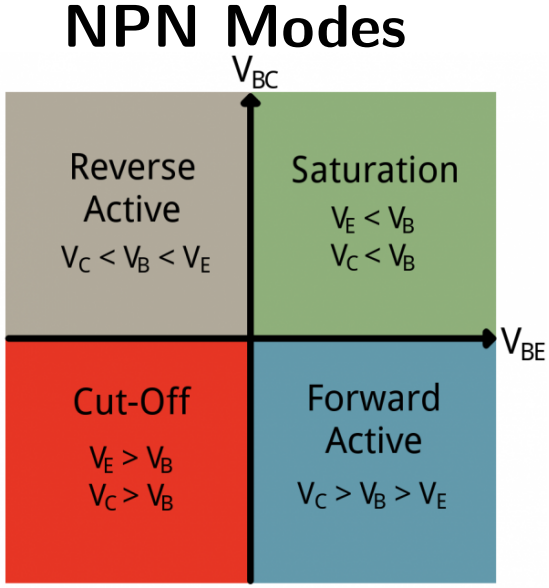
The transistor has three regions, namely base, emitter and collector. The emitter is a heavily doped terminal and emits electrons into the base. The base terminal is lightly doped and passes the emitter-injected electrons on to the collector. The collector terminal is intermediately doped and collects electrons from base. This collector is large as compared with other two regions so it dissipates more heat.



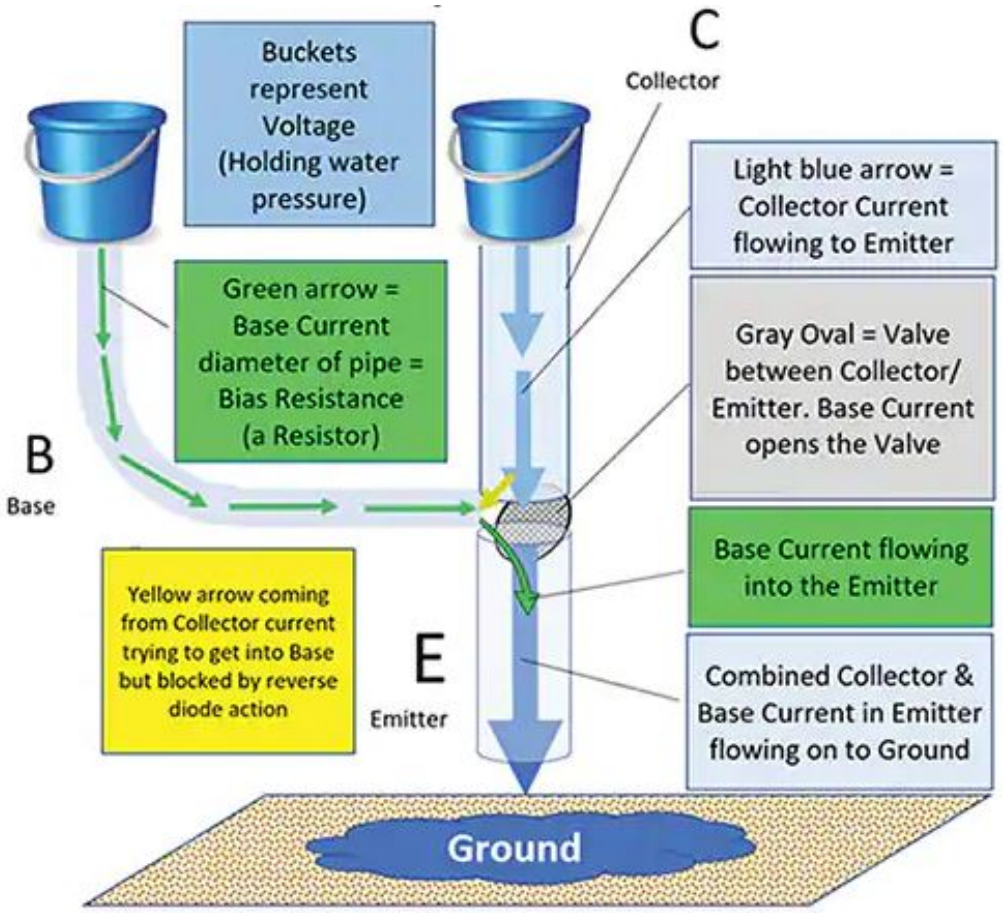
# Bipolar Junction Transistor - Modes of Operation

- **Saturation:** Current freely flows from collector to emitter. (ON Switch)
- **Cut-off:** No current flows from collector to emitter. (OFF Switch)
- **Active:** The current from collector to emitter is proportional to the current flowing into the base. (Amplifier)
- **Reverse-Active:** Like active mode, the current is proportional to the base current, but it flows in reverse from emitter to collector (not the purpose transistors were designed for).



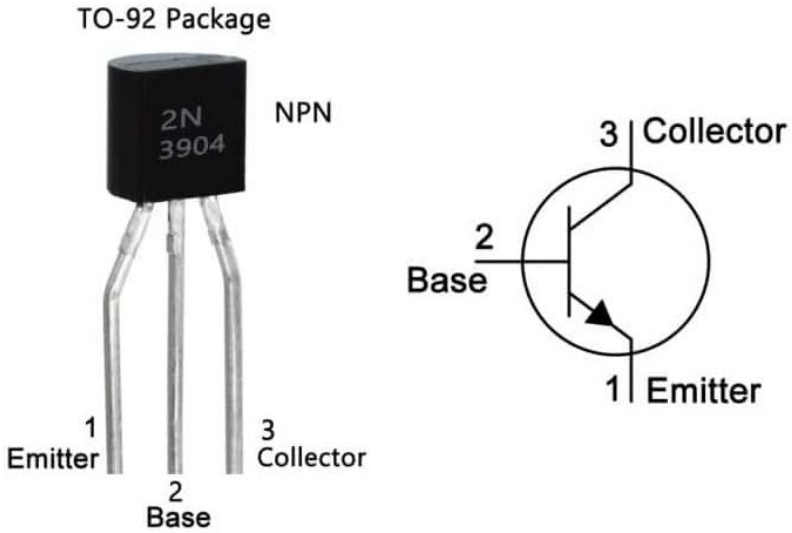

| Voltage relations | NPN Mode | PNP Mode |
|---|---|---|
| $V_E < V_B < V_C$ | Active | Reverse |
| $V_E < V_B > V_C$ | Saturation | Cutoff |
| $V_E > V_B < V_C$ | Cutoff | Saturation |
| $V_E > V_B > V_C$ | Reverse | Active |

# Water Analogy

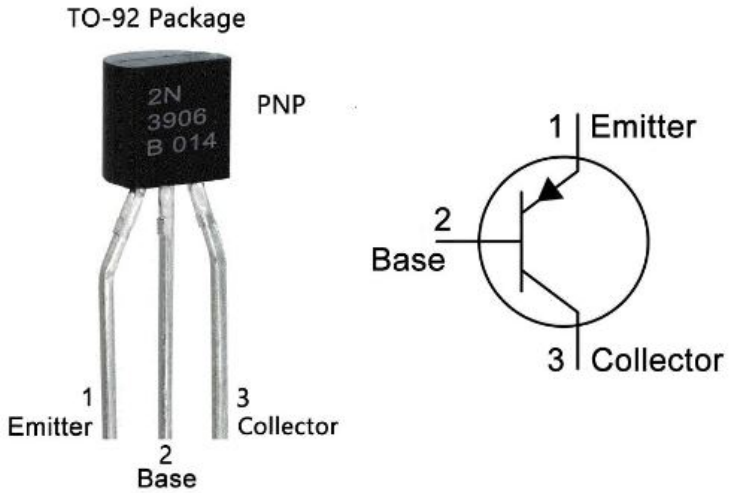


# BJT Transistor Pinouts

## 2N3904 NPN Transistor

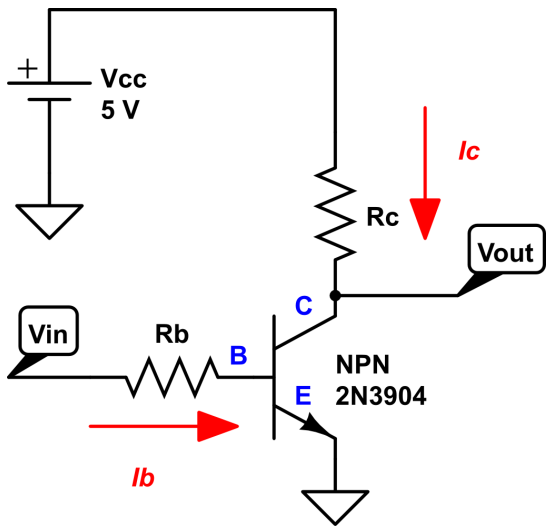



## 2N3906 PNP Transistor



# Active Mode NPN Transistor Circuit

If you apply a voltage $V_{IN}$ that is high enough to forward-bias the base-to-emitter junction, current ($I_B$) will flow from the input terminal, through $R_B$, through the BE junction, to ground. Current ($I_C$) will also flow through $R_C$ and the collector-to-emitter portion of the transistor.

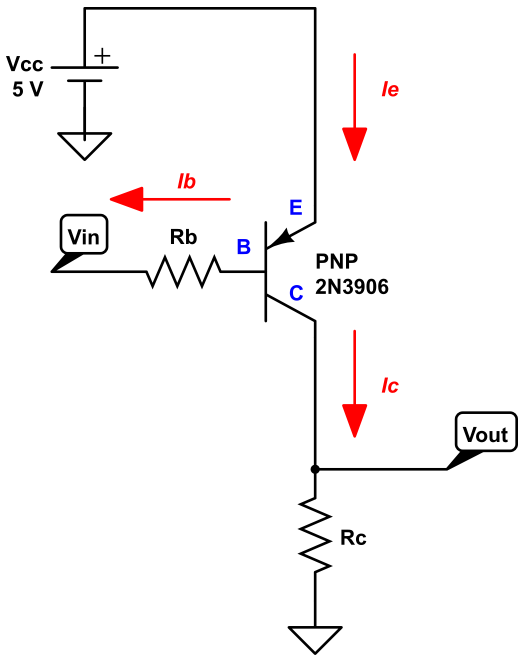



**NOTE:** $V_{OUT}$ is an amplified but inverted signal of $V_{IN}$.
This simple circuit will step-up a 0 - 3.3V output from the microcontroller to 0 - 5.0V (inverted). The low impedance of the output will also provide sufficient current to drive a higher current device (e.g., a relay).

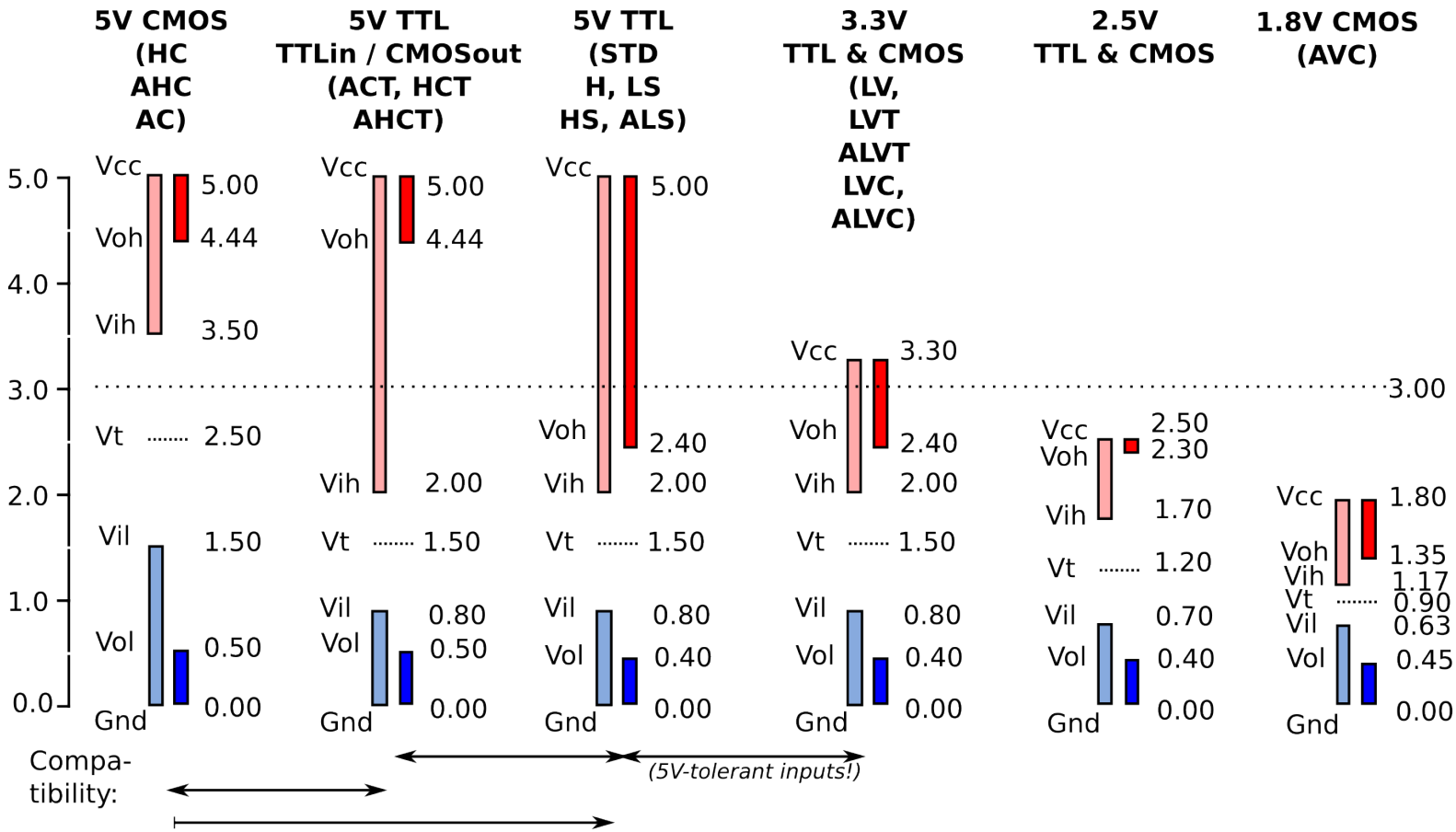
# PNP Transistor

NPN Transistors are more common than PNP for a number of reasons:

- The voltage and current behavior of an NPN transistor is significantly more intuitive.
- When a switch or driver circuit is required, NPNs provide a more straightforward interface to digital output signals (such as a control signal generated by a microcontroller).
- NPNs are higher performance (faster switching speeds) due to higher mobility of electrons vs holes.



# Logic Voltage Level Standards

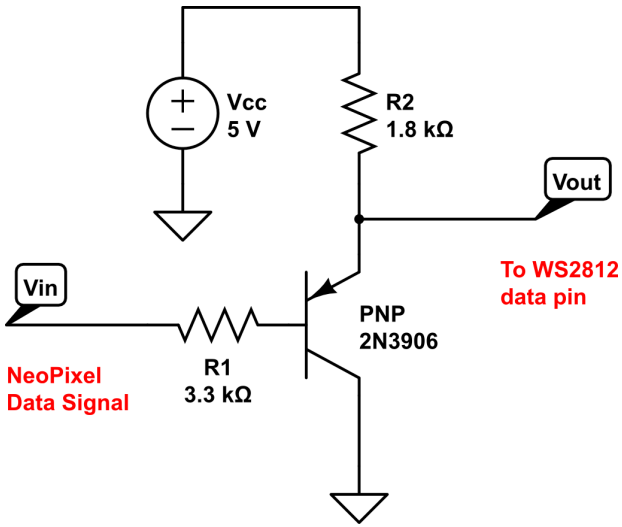



*Data source: EETimes, A brief recap of popular logic standards (Mark Pearson, Maxim).*

Or, what is wrong with the NeoPixels.

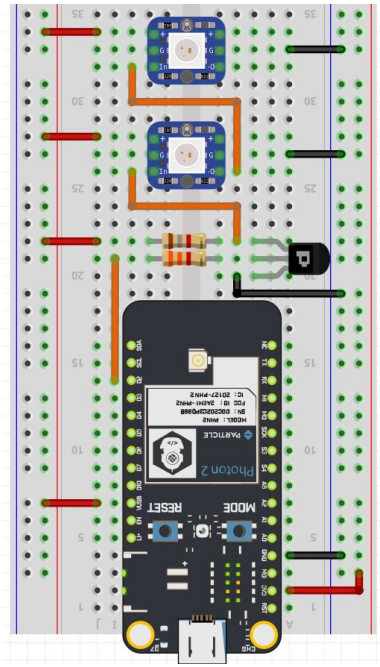
# Emitter Follower - Saturation and Cutoff Mode

- NeoPixels are designed around 5V CMOS transistors.
  - $V_{IH} > 3.5V$
  - 3.3V Microcontroller $V_{OH} = 3.3V$
- An Emitter Follower (i.e., a PNP transistor wired backwards) is a current amplifier, but will also produce a $V_{OUT} = 3.9V$.
- Alternatively, the first NeoPixel could be sacrificed by reducing its $V_{cc}$ to 4.3V with a diode.

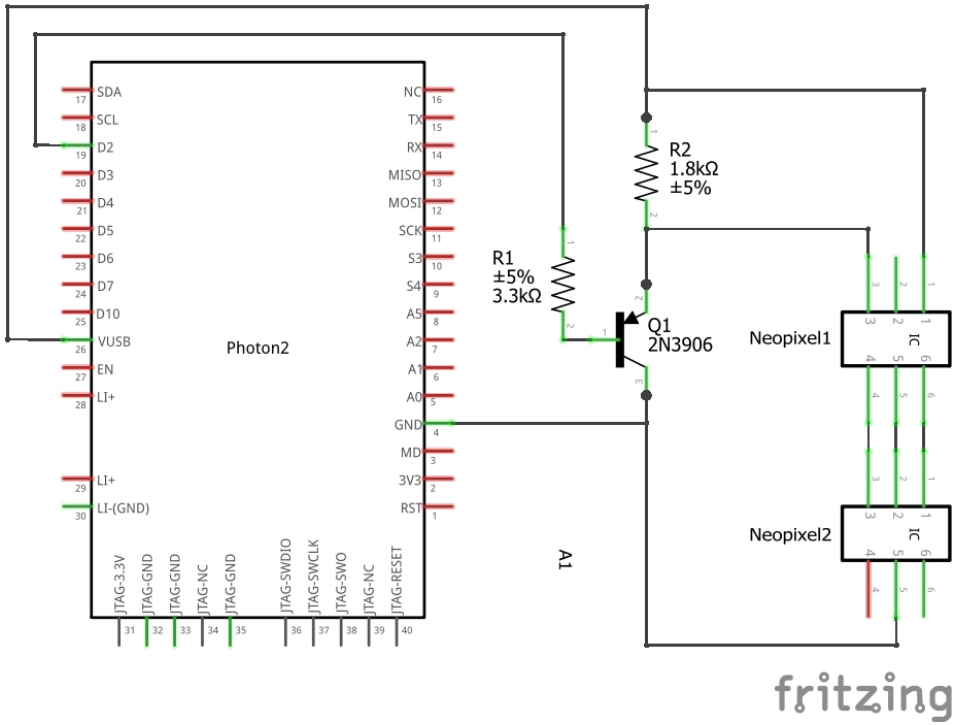


# Emitter Follower Layout (using 2N3906 PNP Transistor)

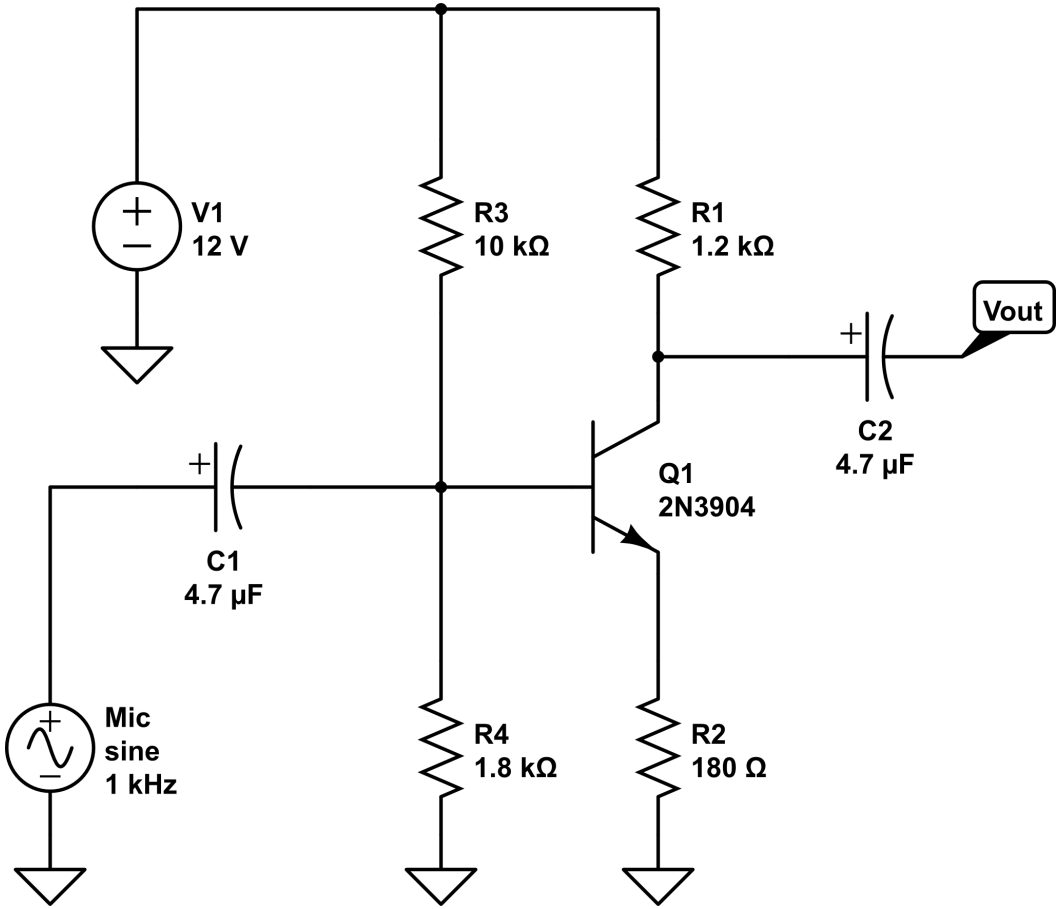
## Fritzing



## Schematic

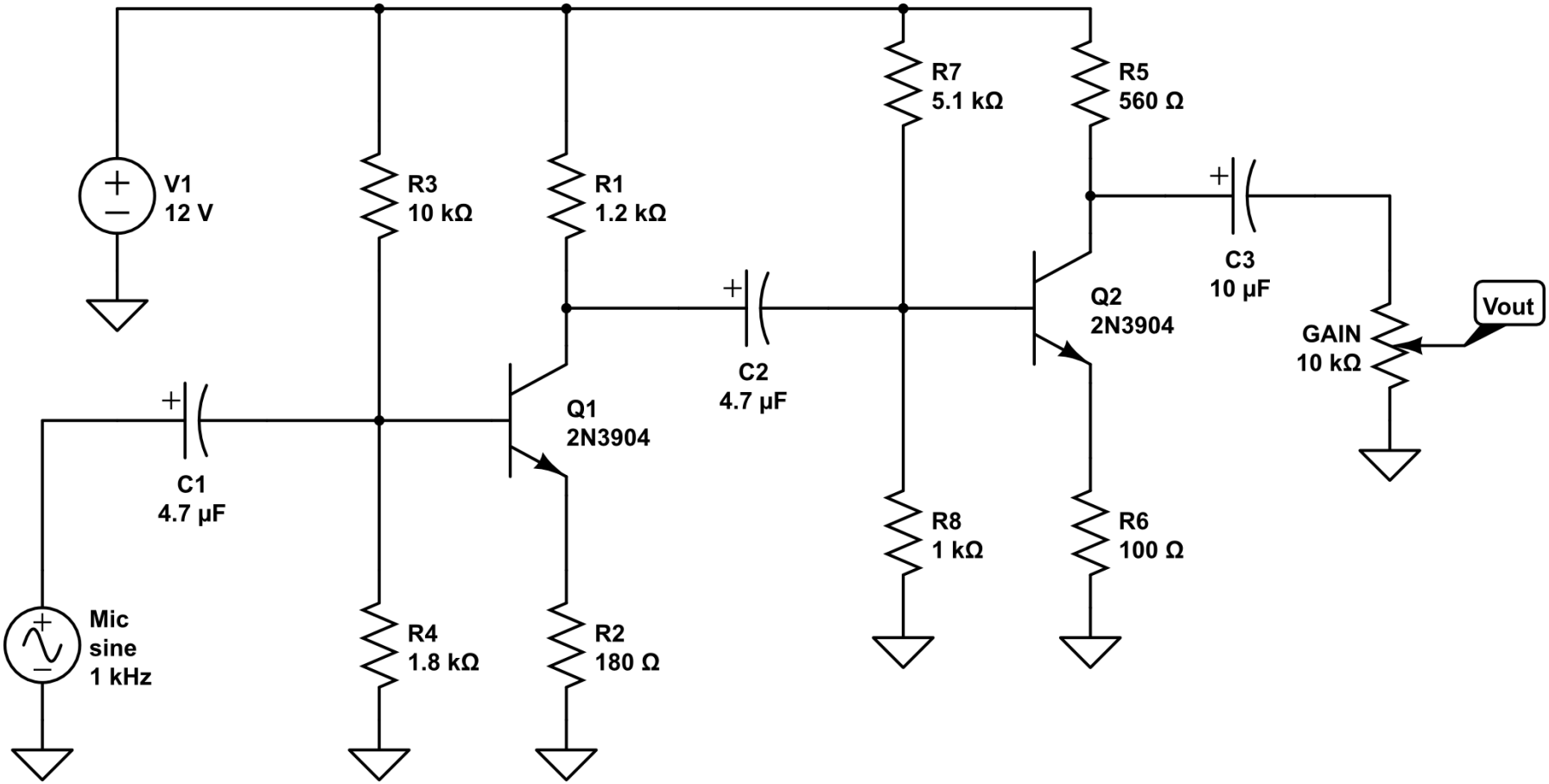


# Typical NPN Pre-Amplifier Circuit

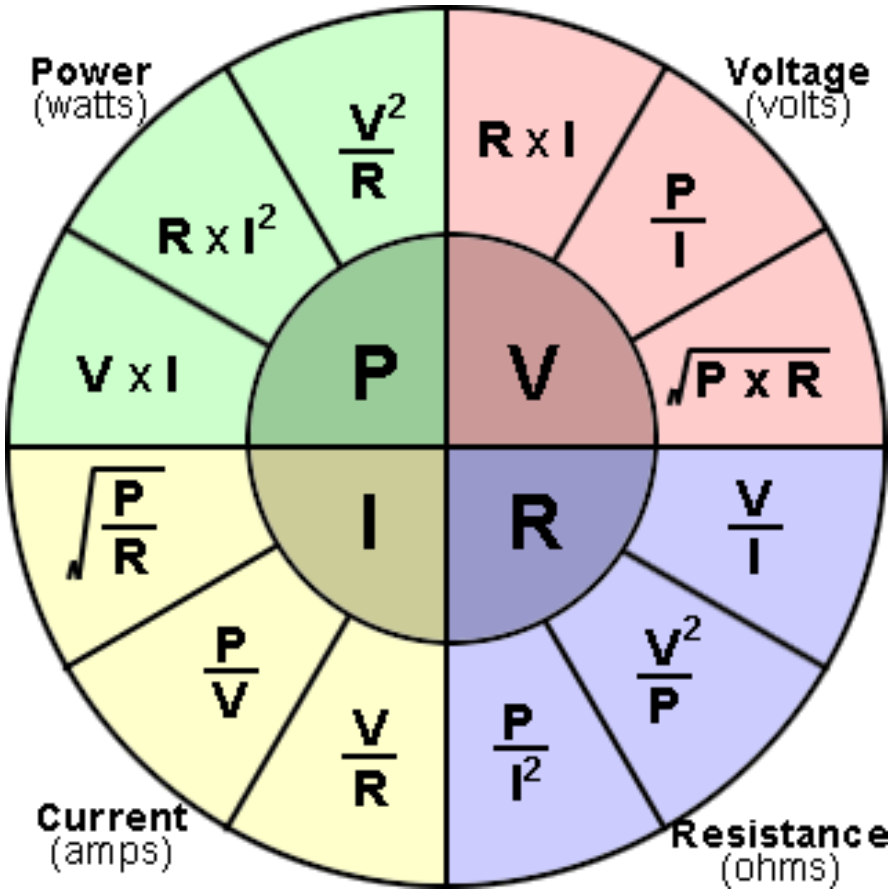



This is referred to as a Common Emitter amplifier as the Emitter ground is common to both the input and output voltage. The Common Emitter amplifies both voltage and current.

# Two Stage Pre-Amplifier



# Ohm's Law - Revisited



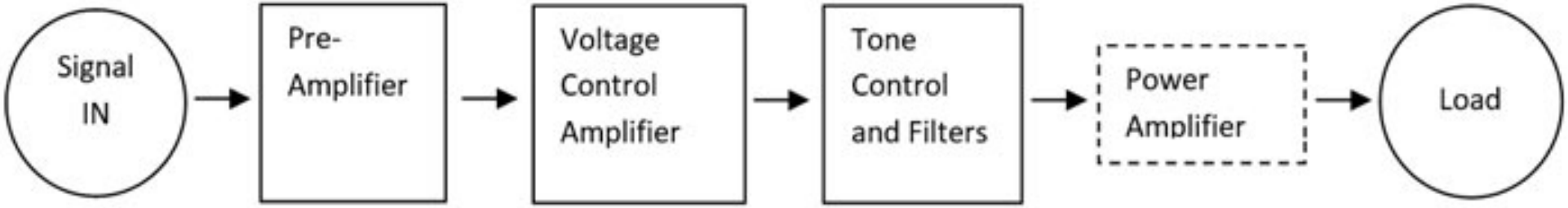
# PreAmp vs PowerAmp

PreAmp:

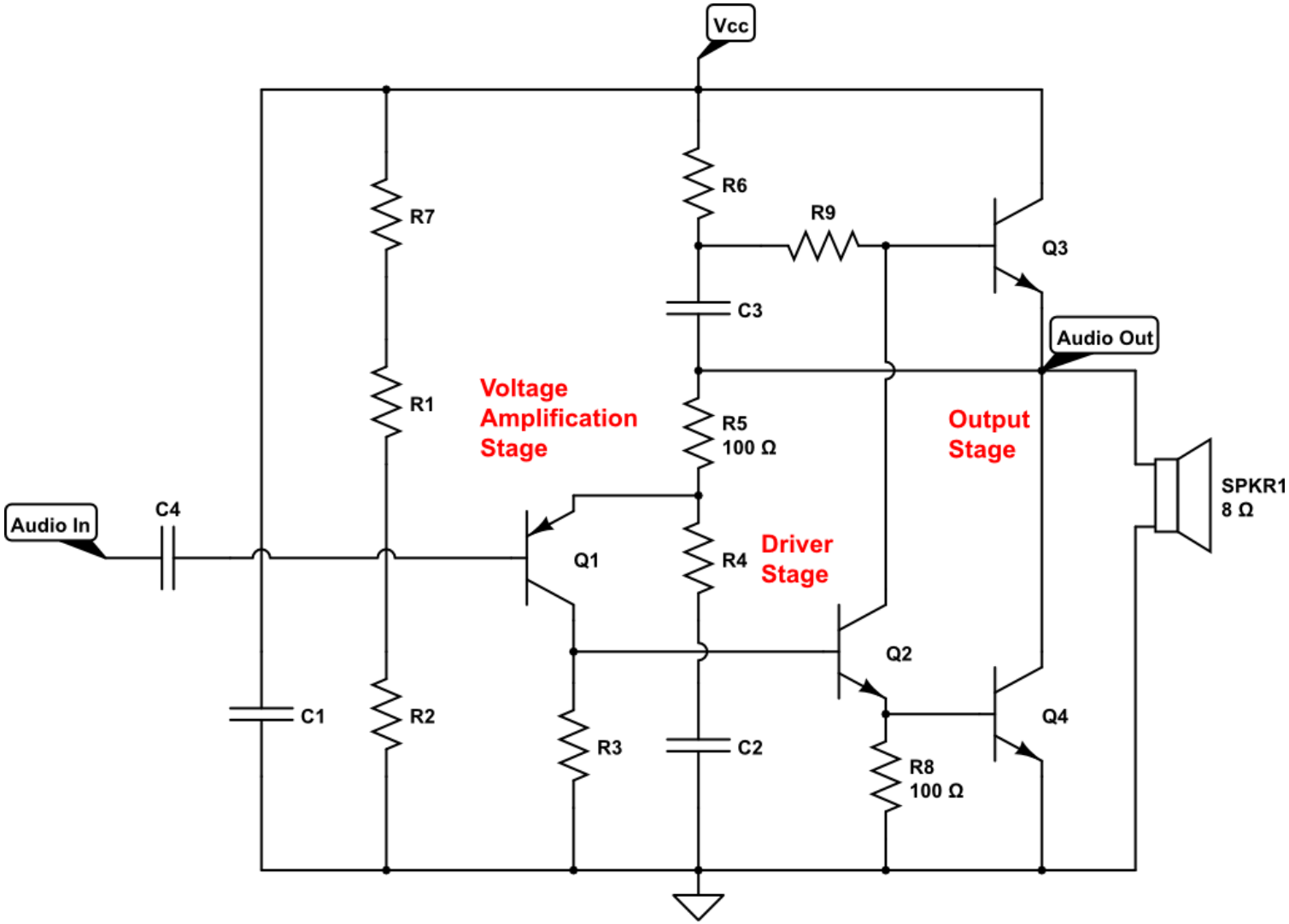
- A preamp boosts the signal up to 'line level'.
- Guitar PreAmp
  - A pure guitar signal typically sounds weak and anaemic, as is seen if a guitar is directly plugged PA system.
  - A preamp is able to raise a guitar's signal up to an audible volume.
  - It can also be used to affect the audio characteristics.

PowerAmp:

- A power amp boosts that line level signal even more – so that it can be projected through speakers.

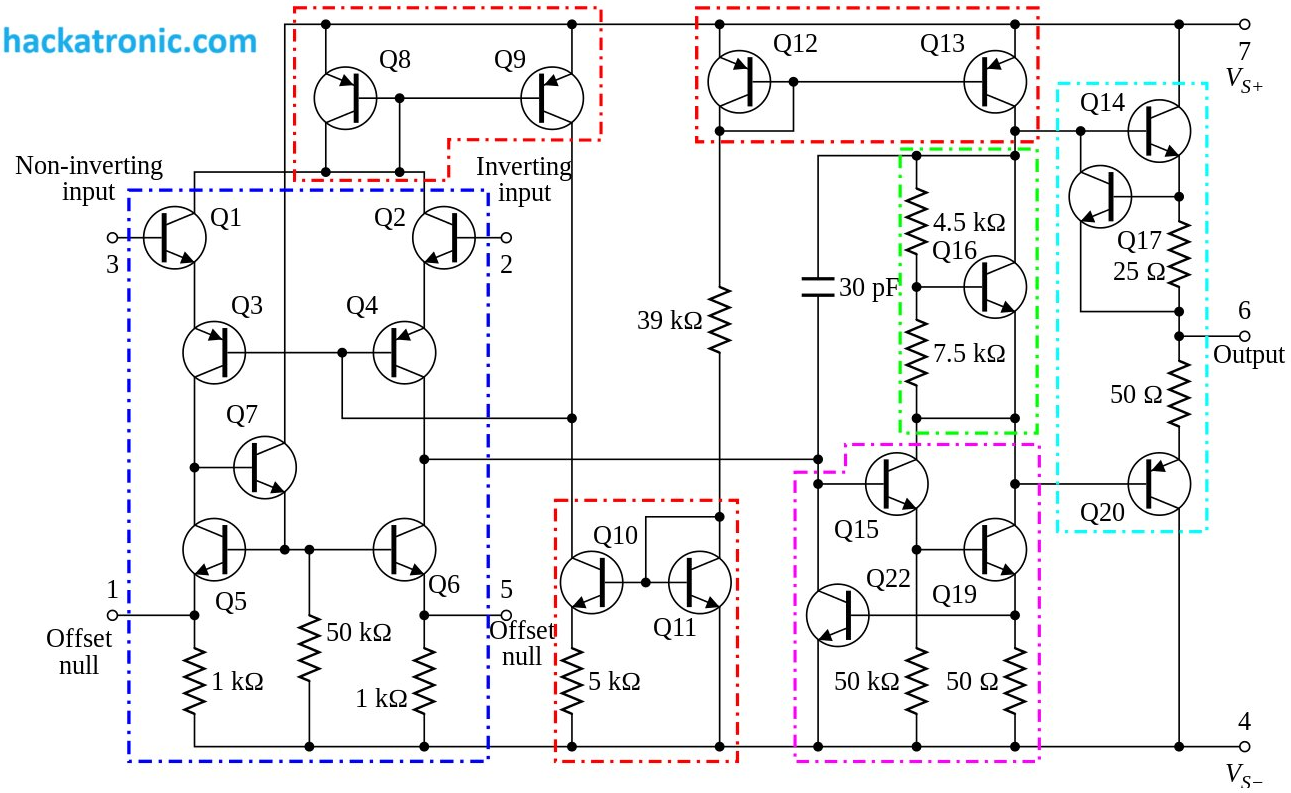


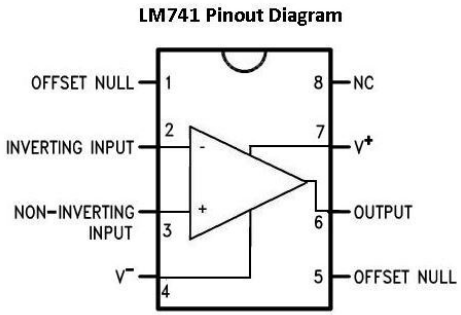
# Power Amplifier



# Operational Amplifiers or Op Amps

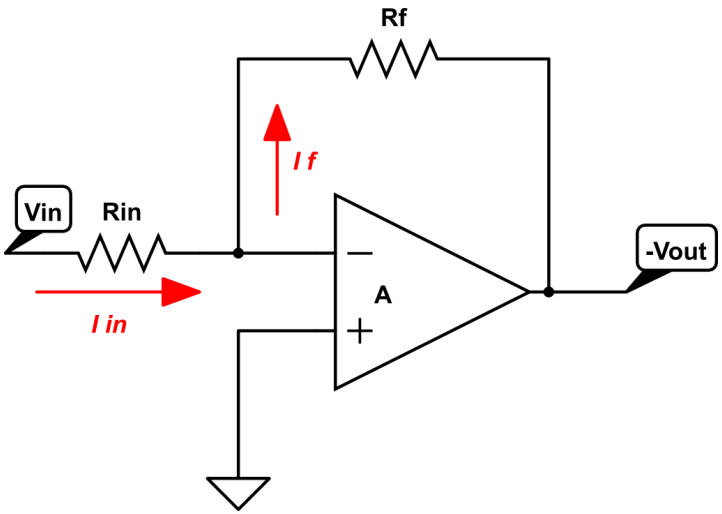
Integrated Circuit amplifier such as the LM741.



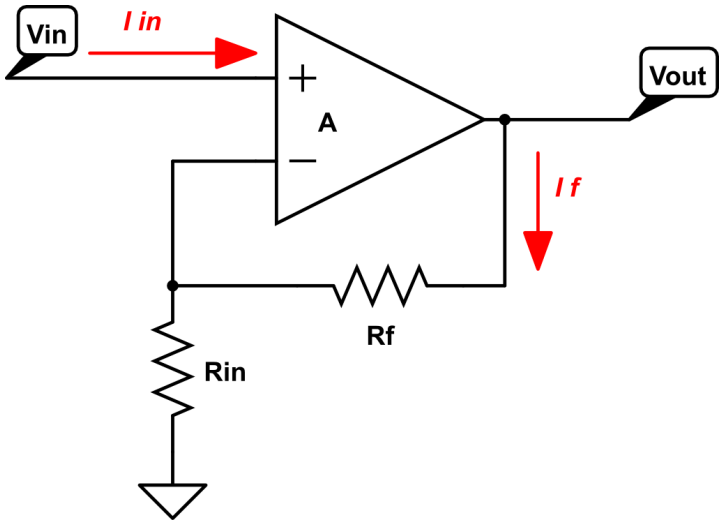
# Op Amp Circuits

## Inverting Op Amp




$$A = \frac{V_{out}}{V_{in}} = -\frac{R_f}{R_{in}}$$

## Non-inverting Op Amp

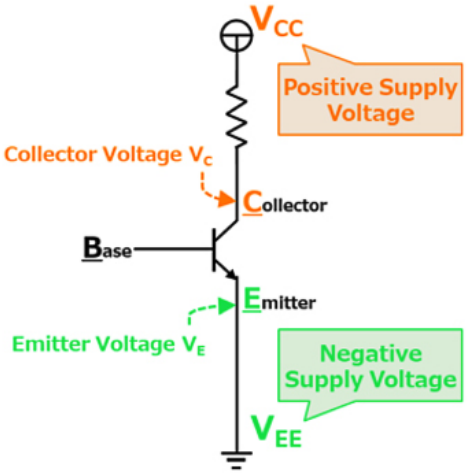



$$A = \frac{V_{out}}{V_{in}} = 1 + \frac{R_f}{R_{in}}$$

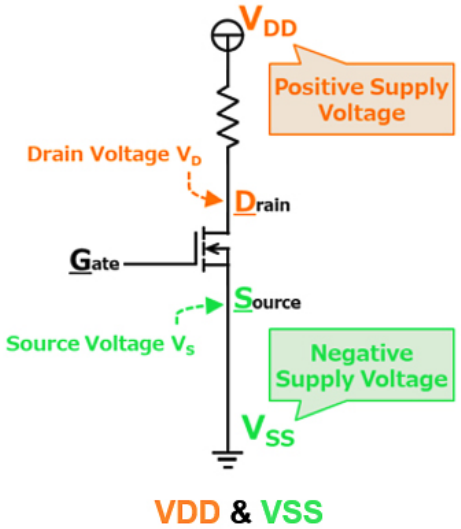
Power the OpAmp with $V^+ = 12V$ and $V^- = 0$

# A word about Supply Voltage Designations

Bipolar Junction Transistors




Metal Oxide Semiconductor Field Effect Transistor (MOSFET)

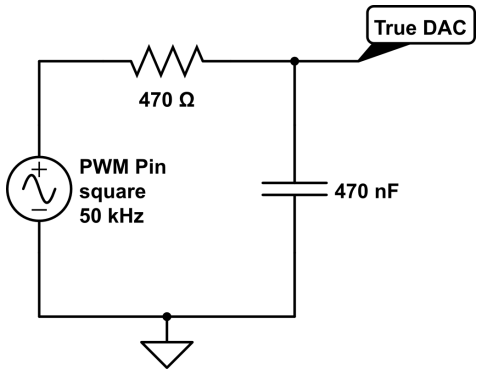


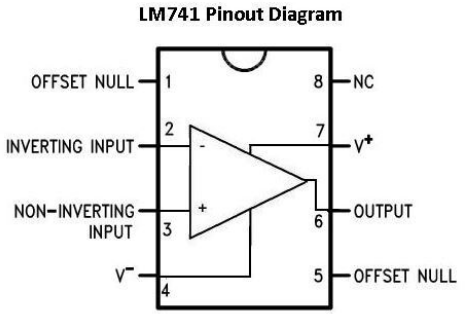

Supply Voltage Designations:

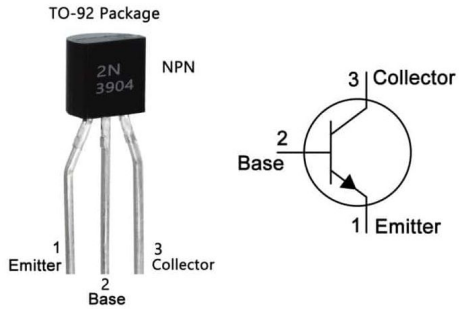
- Positive Supply Voltage: ($V_{cc}$ or $V_{dd}$) is equal to $V_{in}$ or $V^+$.
- Negative Supppy Voltage: ($V_{ss}$ or $V_{ee}$) is equal to $GND$ or $V^-$.

# Assignment: L10_Semiconductor

1. L10_01_NeoPixel
   - Add a Emitter-Follower into your NeoPixel circuit to boost the pixel commands to 5V. Test with L04_01_NeoPixel.
2. L10_02_OpAmp
   - Using the DAC and code from L05_00_lowPass to create a sine wave output (reduce the amplitude from 127.5 to 30).
   - Amplify the True DAC output with a non-inverting LM741 Op Amp. Use a potentiometer for $R_{in}$.
   - Measure the True DAC and Post-OpAmp signals with an oscilloscope.
3. L10_03_NPNAmp
   - Amplify using an NPN preamp instead.
   - Measure the circuit at each node using the oscilloscope.

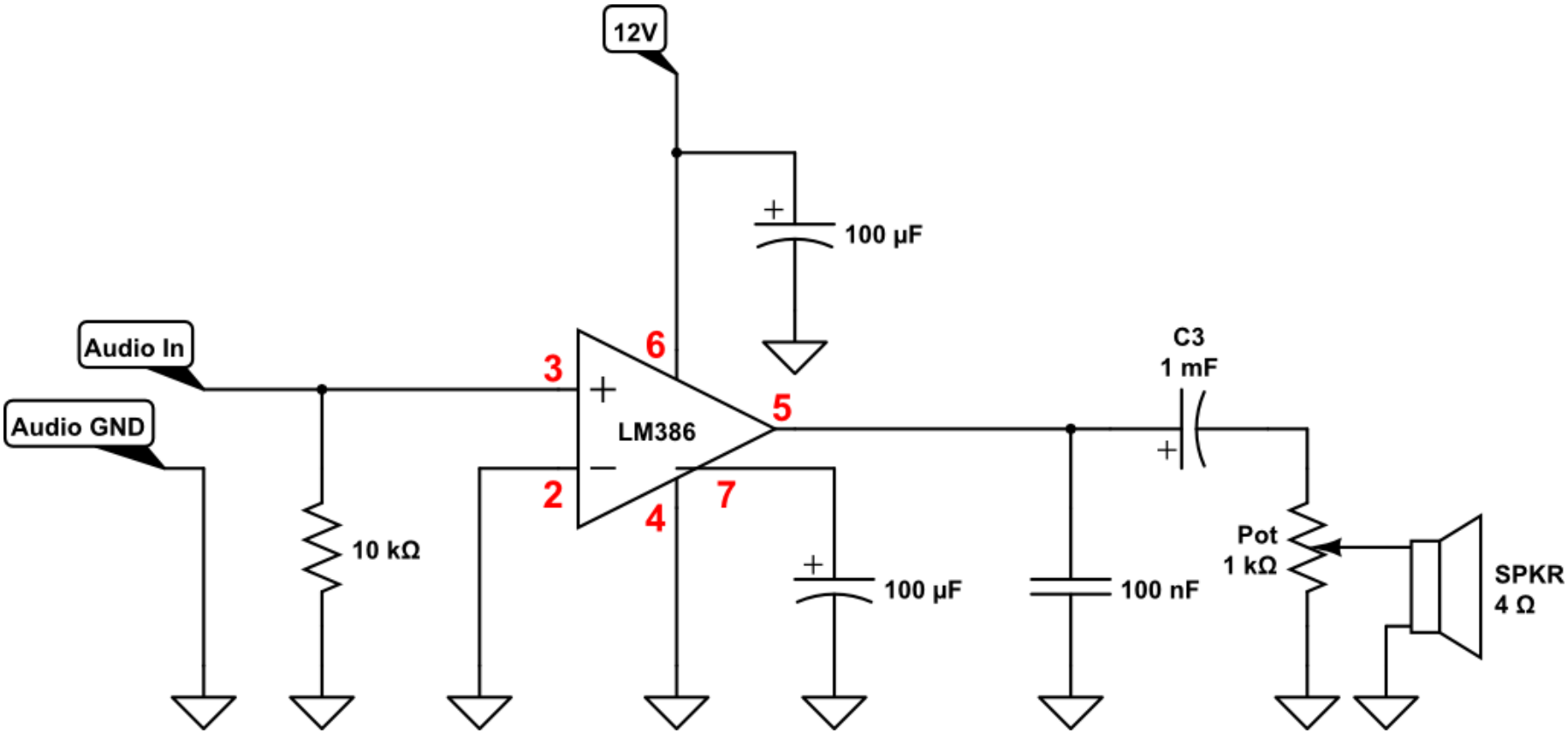


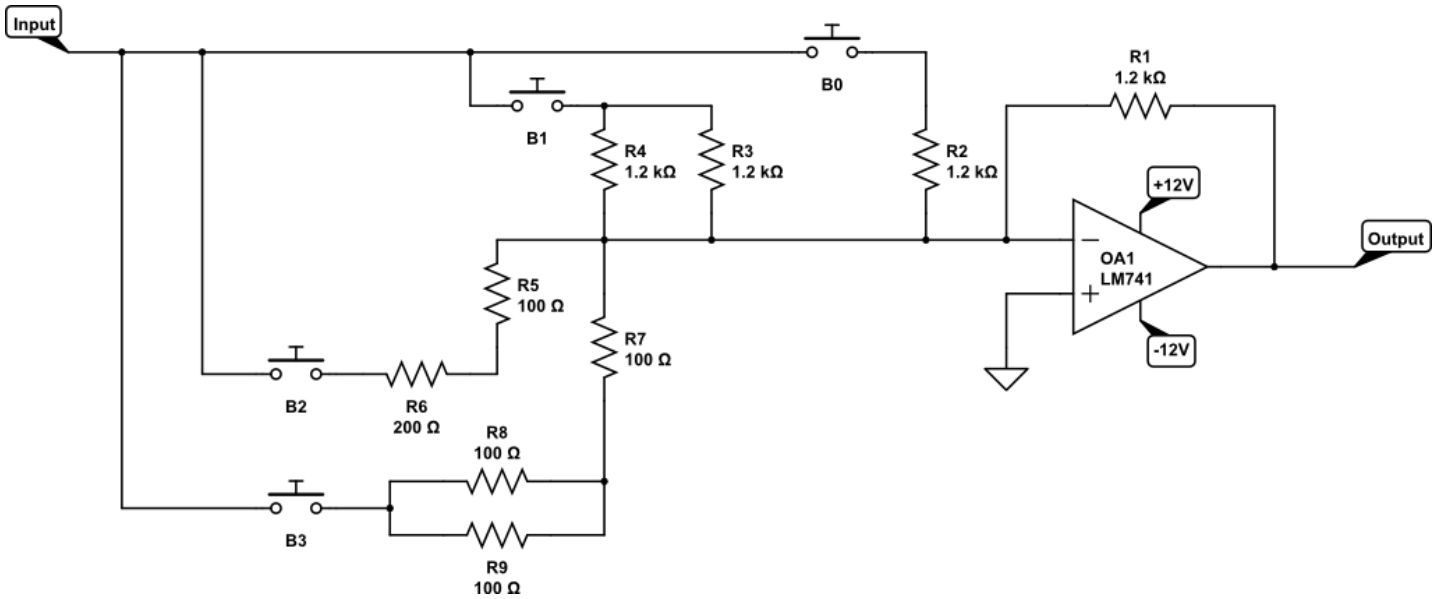
# L10_04_AudioAmp (Extra Credit)

The LM386 is specifically designed for use in audio applications. Its performance is based on driving speakers




- Build the circuit, test with signal generator and oscilloscope.
- Then, hook up sound from phone and speaker

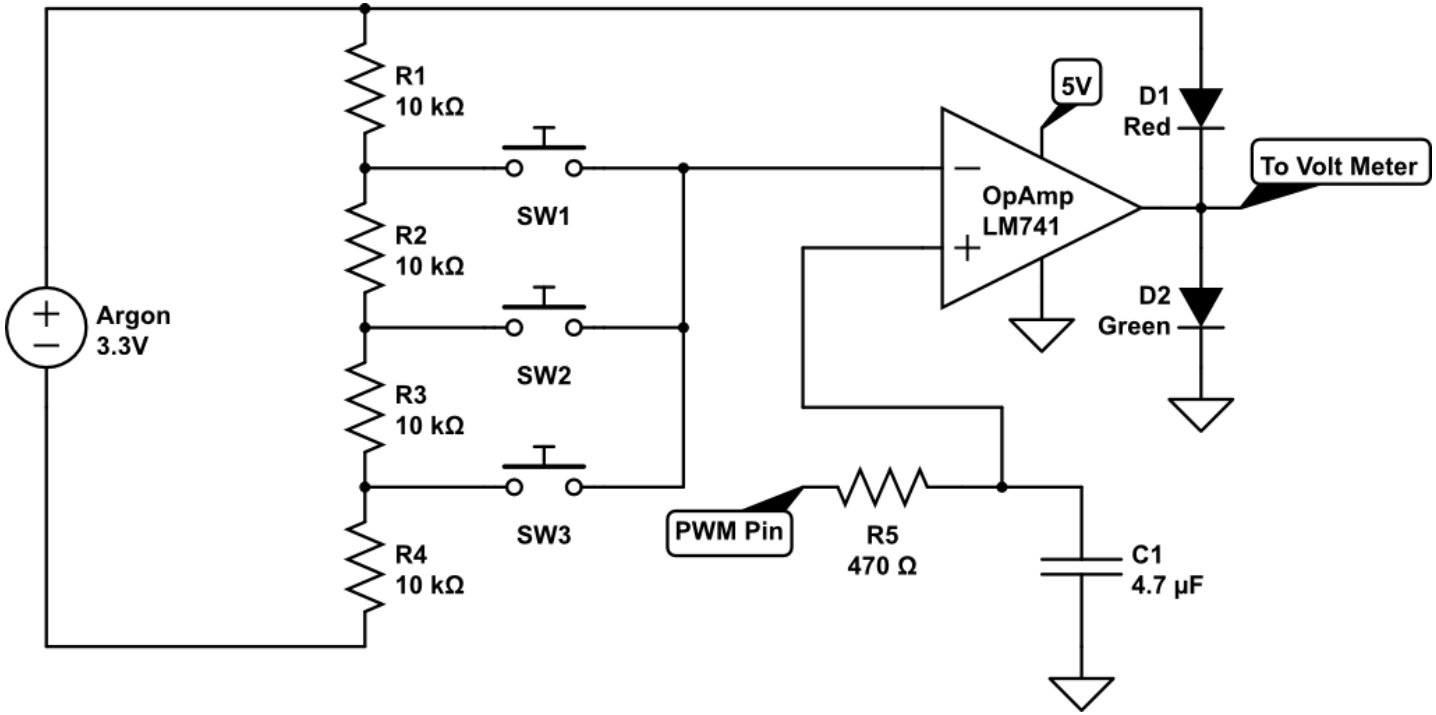
# L10_05_MysteryCircuit (Extra Credit)




- Layout circuit in Fritzing - there is no Particle in this circuit
- Build and test circuit with input at Oscilliscope station
- Record output voltage for all combinations of buttons presses
- Figure out what it is doing and why it works.

Power the OpAmp with $V^+ = 12V$ and $V^- = -12V$

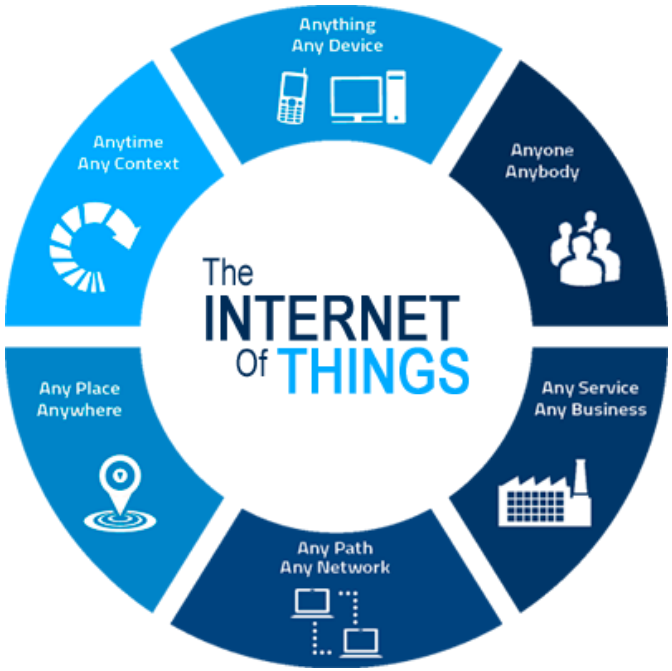
# L10_06_MysteryCircuit2 (Extra Credit)




- Set analogWrite frequency to 50k
- AnalogWrite 1.8V and see the result of different button presses.
- Try different analogWrite voltages

Power the OpAmp with $V^+ = 12V$ and $V^- = 0V$

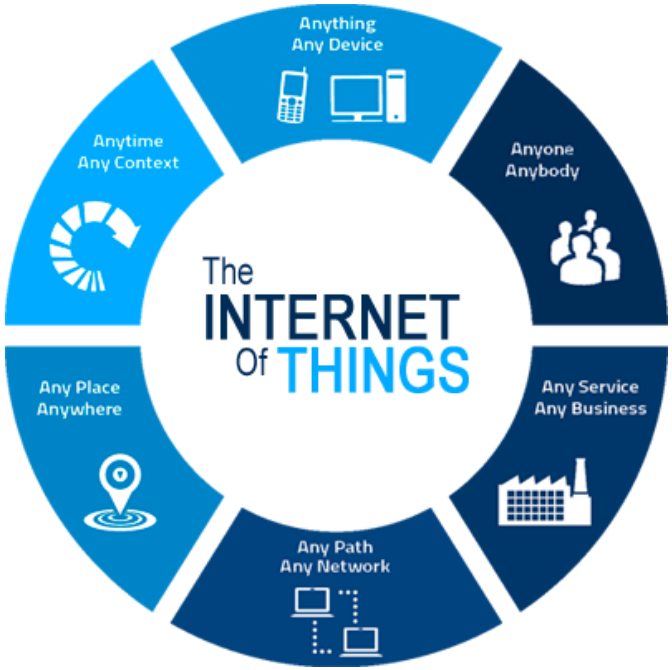
# Module 10 Review




- Learning Objectives
  1. Semiconductors
  2. Diodes
  3. Transistors
  4. Amplifiers

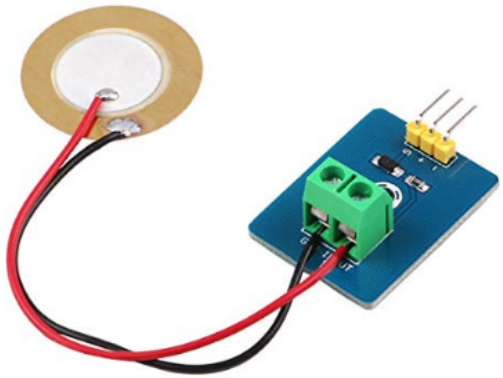
# Module 11 - Sensors

# Module 11 Objectives




- Learning Objectives
  1. Piezoelectric Elements
  2. Load Cells
  3. Moisture Sensors
  4. New sensors - learning on your own
  5. String datatype
- Additional Items
  1. 3D Modeling Lesson 5 - IoT Case
  2. Quiz 7

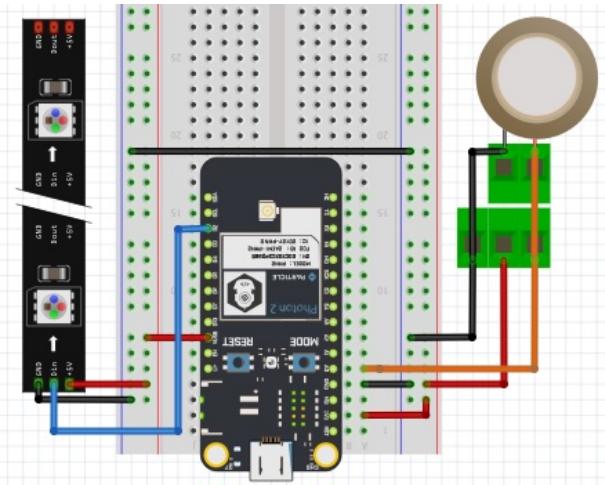
# Piezoelectric Elements



- The piezoelectric effect is the creation of electrical potential (voltage) across the side of a crystal when subject to mechanical stress.
- Conversely, a crystal becomes mechanically stressed (deformed in shape) when a voltage is applied across opposite faces.
- By utilizing an analogRead(), the voltage (and thus the amount of pressure on the crystal) can be measured.

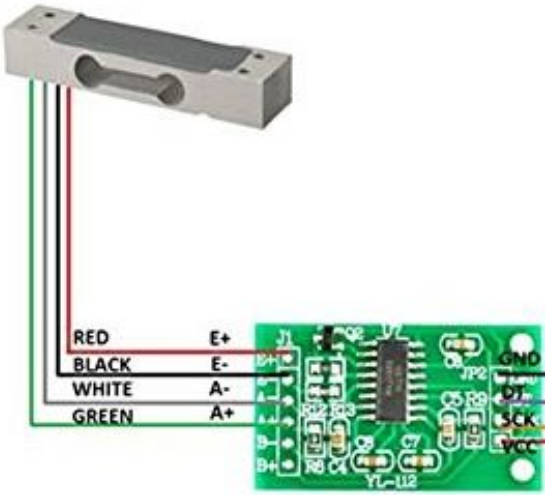
# Assignment: Carnival Game

1. L11_01_HighStriker
   - Connect the piezo sensor and the NeoPixel tower to the Particle
   - Each time the piezo is struck, find the maximum voltage generated.
   - Light up the NeoPixel tower proportional to the force the piezo is struck with.
   - Small delay ($< 150ms$) is allowed to create the moving effect as the tower lights up.



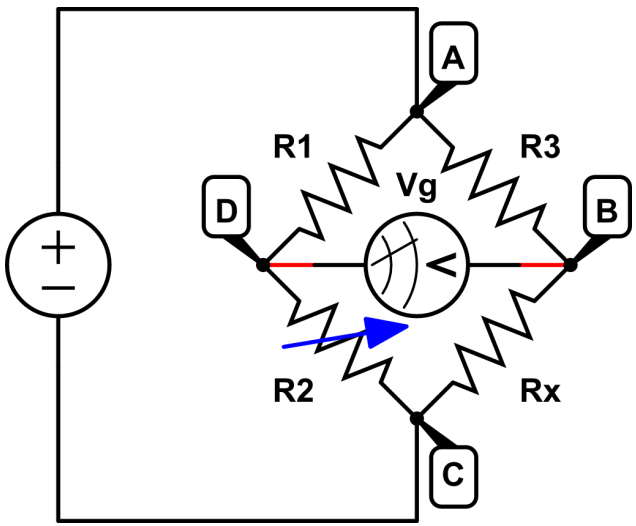
- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

# Load Cells




1. A load cell is a force transducer. It converts a force such as tension, compression, pressure, or torque into an electrical signal that can be measured and standardized. As the force applied to the load cell increases, the electrical signal changes proportionally. The most common types of load cells used are hydraulic, pneumatic, and strain gauge.
2. The HX711 module is a precision 24-bit analog-to-digital converter (ADC) designed for weigh scales and industrial control applications to interface directly with a bridge sensor.

# Wheatstone Bridge




1. The Wheatstone bridge was invented by Samuel Hunter Christie in 1833 and improved and popularized by Sir Charles Wheatstone in 1843.
2. A Wheatstone bridge is an electrical circuit used to measure an unknown electrical resistance by balancing two legs of a bridge circuit, one leg of which includes the unknown component.
3. The primary benefit of the circuit is its ability to provide extremely accurate measurements (in contrast with something like a simple voltage divider).

# HX711A Library - Linear Conversion

The Load Cell Library (HX711A) will convert the input into everyday units.

- tare(): The zero'ing function will automatically set the OFFSET
- CALFACTOR: This is used to convert from the 24-bit integer to units of your choice (grams, pounds, tons, etc.)

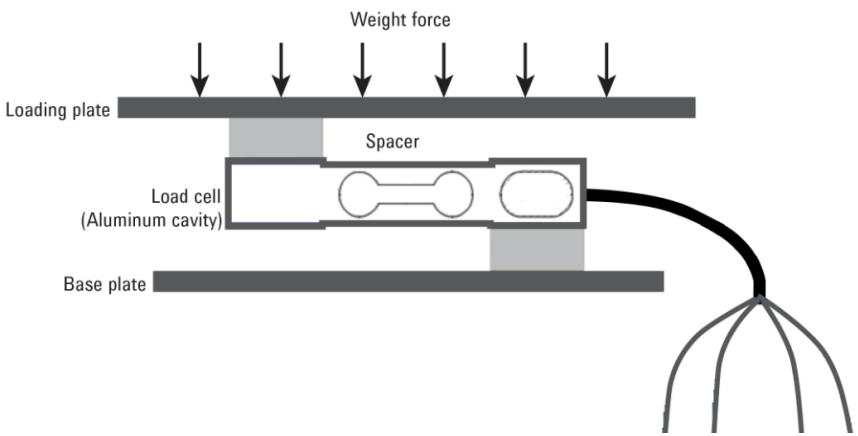
The HX711 Class will automatically convert the measured value (minus the OFFSET) to desired units using CALFACTOR:

$$units = \frac{1}{CALFACTOR} * (value - OFFSET)$$

# HX711A Library

```
// From the Command Palette install the HX711A library, that will give you HX711.h
#include "HX711.h"
HX711 myScale(DT,CLK);     // any two digital pins

const int CALFACTOR=1000; //changing value changes get_units units (lb, g, ton, etc.)
const int SAMPLES=10; //number of data points averaged when using get_units or get_value

float weight, rawData, calibration;
int offset;

void setup() {
  myScale.set_scale();            // initialize loadcell
  delay(5000);                    // let the loadcell settle
  myScale.tare();                 // set the tare weight (or zero)
  myScale.set_scale(CALFACTOR); //adjust when calibrating scale to desired units
}

void loop() {
  // Using data from loadcell
  weight = myScale.get_units(SAMPLES); // return weight in units set by set_scale();
  delay(5000)                          // add a short wait between readings

  // Other useful HX711 methods
  rawData = myScale.get_value(SAMPLES);// returns raw loadcell reading minus offset
  offset = myScale.get_offset();       // returns the offset set by tare();
  calibration = myScale.get_scale();   // returns the cal_factor used by set_scale();
}
```

# Assignment: L11_Sensor (Learn to Calibrate)




- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L11_02_Scale
   - The LoadCell must be cantilevered similar to the diagram. Using the Epilog Laser or the Wood/Metal shop, create a method to cantilever your scale.
   - Set initial CALFACTOR to 1000 and measure a known weight. (Note: one cup of water (in a paper cup) is approx. 244 g).
   - Adjust CALFACTOR until you get the expected measurement in grams.
   - Post data to Adafruit.io
   - Extra Credit: Send text via IFTTT. There is an Adafruit → IFTTT example in Class_Materials.

# More DataTypes - Strings, strings, and char[]

```
// A string (lowercase 's') is an array of characters
  char lastName[7] = "Rashap";
  char firstName[6] = {'B','R','I','A','N'};
  char name[12];

  //The "*" indicates a pointer, which we will learn about later
  char *myName = "Brian";

// A String is a Class that holds a character array
  String instructor = "BRIAN RASHAP";

void setup() {
  Serial.begin();

  Serial.printf("lastName = %s, %i\n",lastName,sizeof(lastName));
  Serial.printf("firstName = %s, %i\n",firstName,sizeof(firstName));
  Serial.printf("name = %s, %i\n",name,sizeof(name));
  Serial.printf("myName = %s, %i\n",myName,sizeof(myName));

  // We can not use %s for the variable instructor, why?
}
```

```
Serial monitor opened successfully:
lastName = Rashap, 7
firstName = BRIAN, 6
name = , 12
myName = Brian, 4
```

# Too Much Time On My Hands

When the Particle Photon2 connects to the Particle Cloud, it synchronizes its clock to the current time.

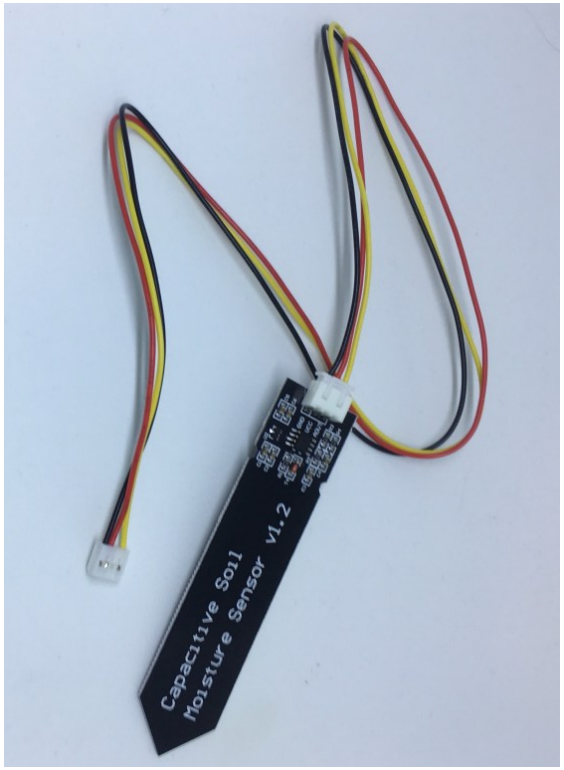
```
// Declare Global Variables in Header
String dateTime, timeOnly;
unsigned int lastTime;

void setup() {
  Time.zone(-7);            // MST = -7, MDT = -6
  Particle.syncTime();      // Sync time with Particle Cloud
}
void loop() {
  dateTime = Time.timeStr();            //Current Date and Time from Particle Time class
  timeOnly = dateTime.substring(11,19); //Extract the Time from the DateTime String
  if(millis()-lastTime>10000) {
    lastTime = millis();

    //%s prints an array of char
    //the .c_str() method converts a String to an array of char
    Serial.printf("Date and time is %s\n",dateTime.c_str());
    Serial.printf("Time is %s\n",timeOnly.c_str());
  }
}
```

To learn more about the String Class: https://docs.particle.io/reference/device-os/api/string-class/string/

# Soil Moisture Sensors

## Resistive Sensor


## Capacitive Sensor

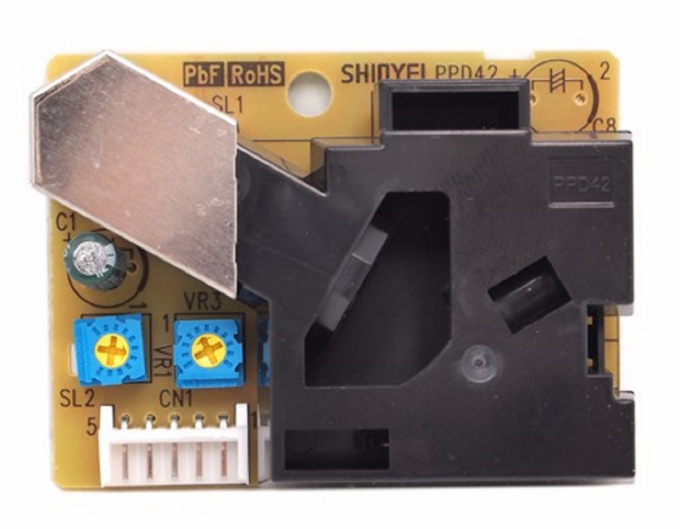


# Assignment: Moisture Probe

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L11_03_Moisture
   - Using the Capacitive Soil Moisture probe, in your notebook note the moisture readings when:
     - Empty Cup
     - Submerged in water to the notch
     - Dry Soil
     - Soil after watered
   - Display the moisture to the OLED with a Time-stamp.

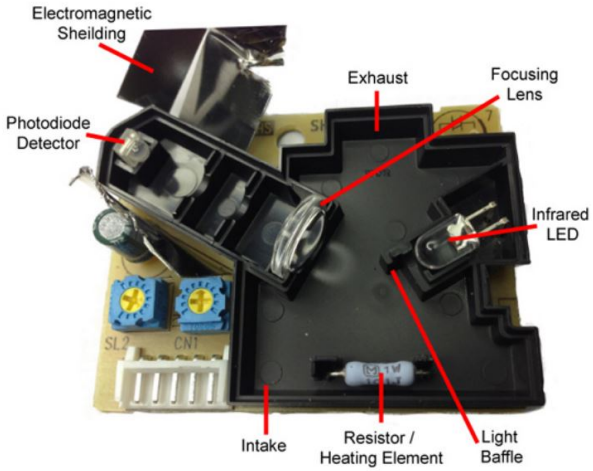
# Seeed Sensors

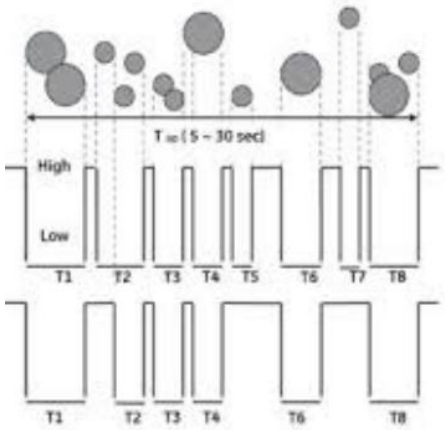


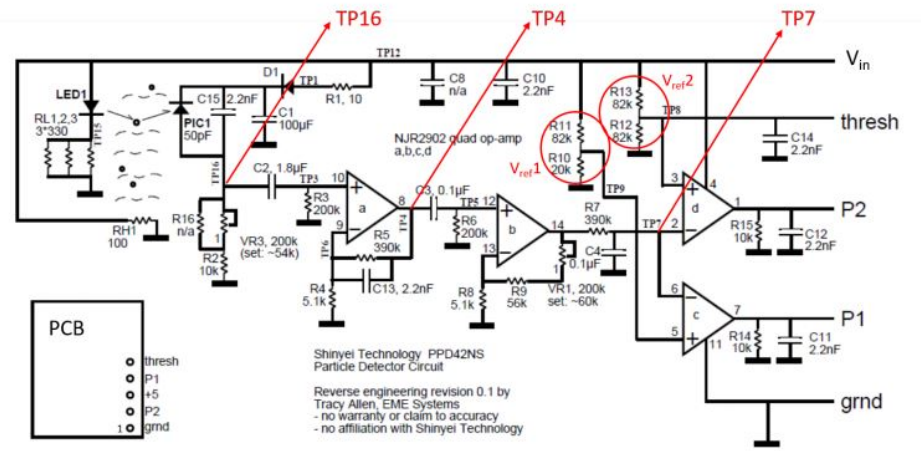

Seeed Grove - Dust Sensor

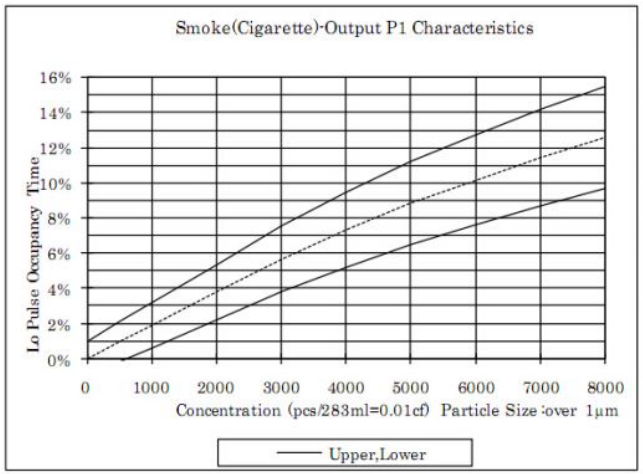
Seeed Grove - Air Quality Sensor v1.3

# Shinyei PPD42NS low-cost dust sensor

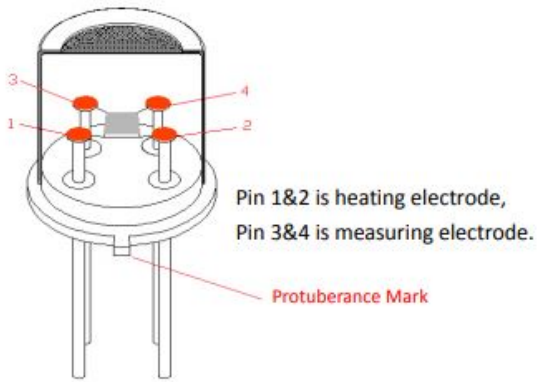


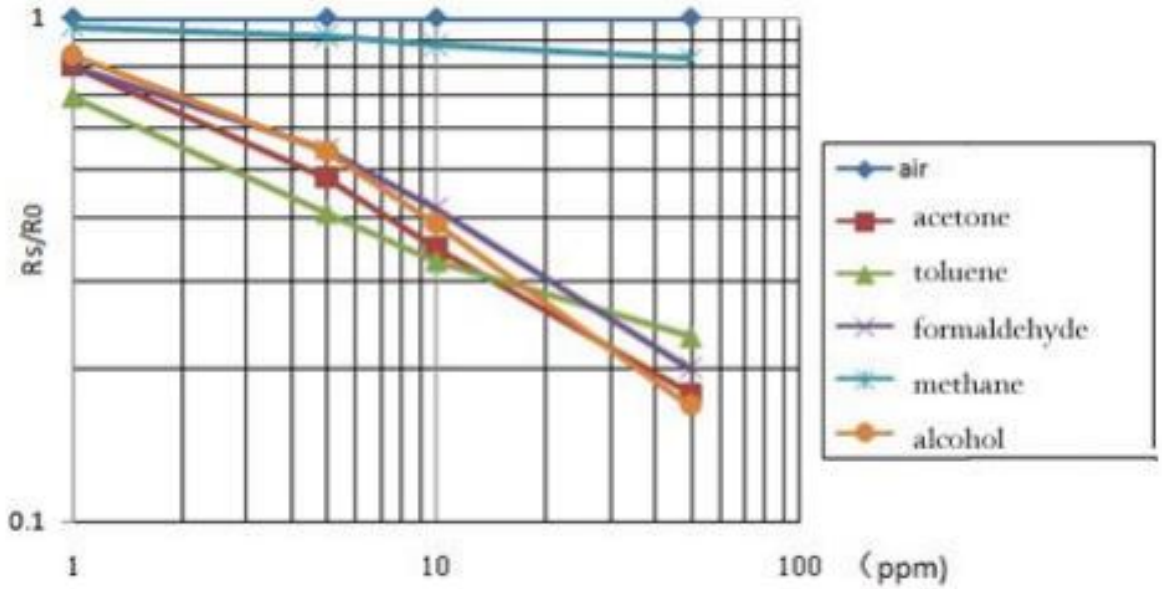



# MP-503 Air Quality Sensor



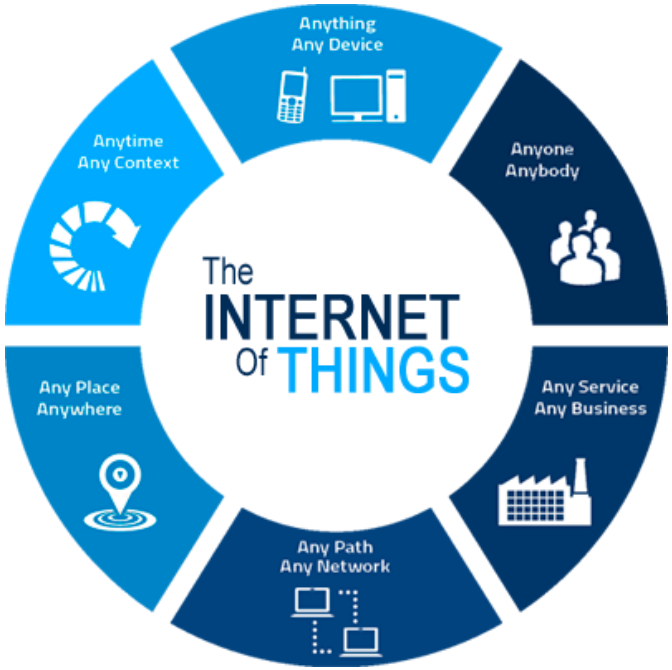



# Seeed Assignment

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L11_04_SeeedSensors
   - Look up the Seeed sensors online to see how they work.
   - Do not blindly copy the examples. Only use the code you need.
   - By looking at the .cpp code, determine how to get a quantitative value for air quality, in addition to the qualitative level.
   - Display air quality and particulate concentration to an Adafruit.io dashboard.

# Module 11 Review




- Learning Objectives
  1. Piezoelectric Elements
  2. Load Cells
  3. Moisture Sensors
  4. New sensors - learning on your own
  5. String datatype
- Additional Items
  1. 3D Modeling Lesson 5 - IoT Case
  2. Quiz 7

# Midterm 2 - House Plant Watering System

# Not Everything is About Voltage

There is often a focus on voltage, it is possible to be current limited.

| Parameter | Symbol | Min | Typ | Peak | Unit |
|---|---|---|---|---|---|
| Operating Current (uC on, peripherals and radio disabled) | $I_{idle}$ | 21.4 | 23.2 | 23.8 | mA |
| Operating Current (uC on, BLE advertising) | $I_{ble\_adv}$ | 54.7 | 58.7 | 70.7 | mA |
| Operating Current (uC on, radio connected to access point) | $I_{wifi\_conn\_ap}$ | 54.6 | 60.5 | 265 | mA |

The $3.3V$ regulator can supply $500mA$. Typical power consumption:

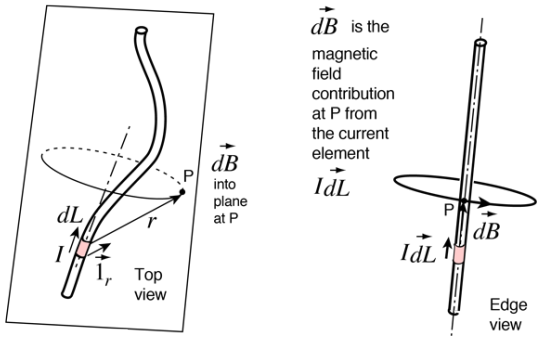
- The Photon2 connected to Wifi can draw up to $265mA$.
- Each GPIO used as an output can supply $4 - 12mA$.
- Each NeoPixel uses up to $60mA$ at full brightness.
- Seeed sensors: dust - $90mA$, AQ sensor - $60mA$
- Plant Watering Pump: $130mA$

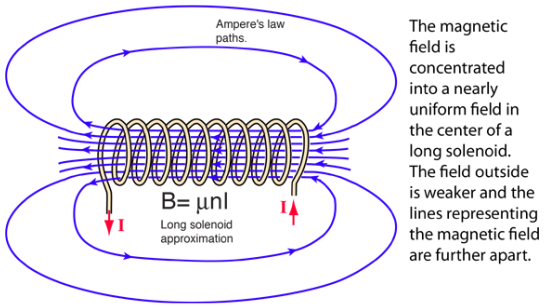
$V_{USB}$ can supply[1] up to $1500mA$[2]. External supply can add more current.

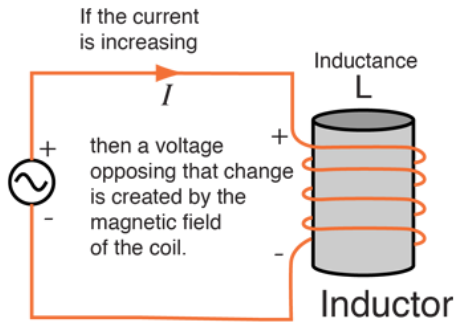
[1] It is all about power ($P = V * I$), so $1500mA$ at $5V$ is $2250mA$ at $3.3V$

[2] Specific amount depends on type of USB port used (2.0 vs 3.x vs USB-C)

# Inductors

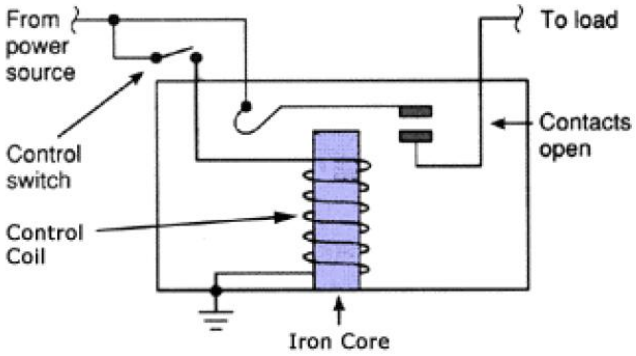


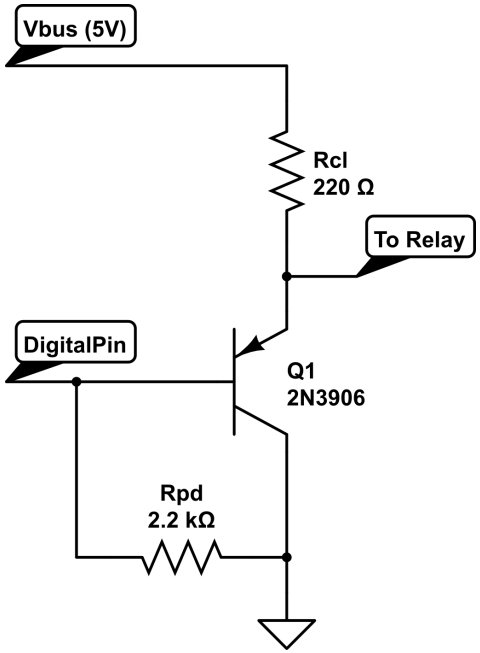




- Current flowing in a wire produces a magnetic field (B) around the wire (from Ampere's Law).
- Wire wrapped into a coil produces a magnetic field that resembles a bar magnet through the center of the coil.
- Also, in a coil, this magnetic field produces an effect known as Inductance (L) that opposes changes in electric current.

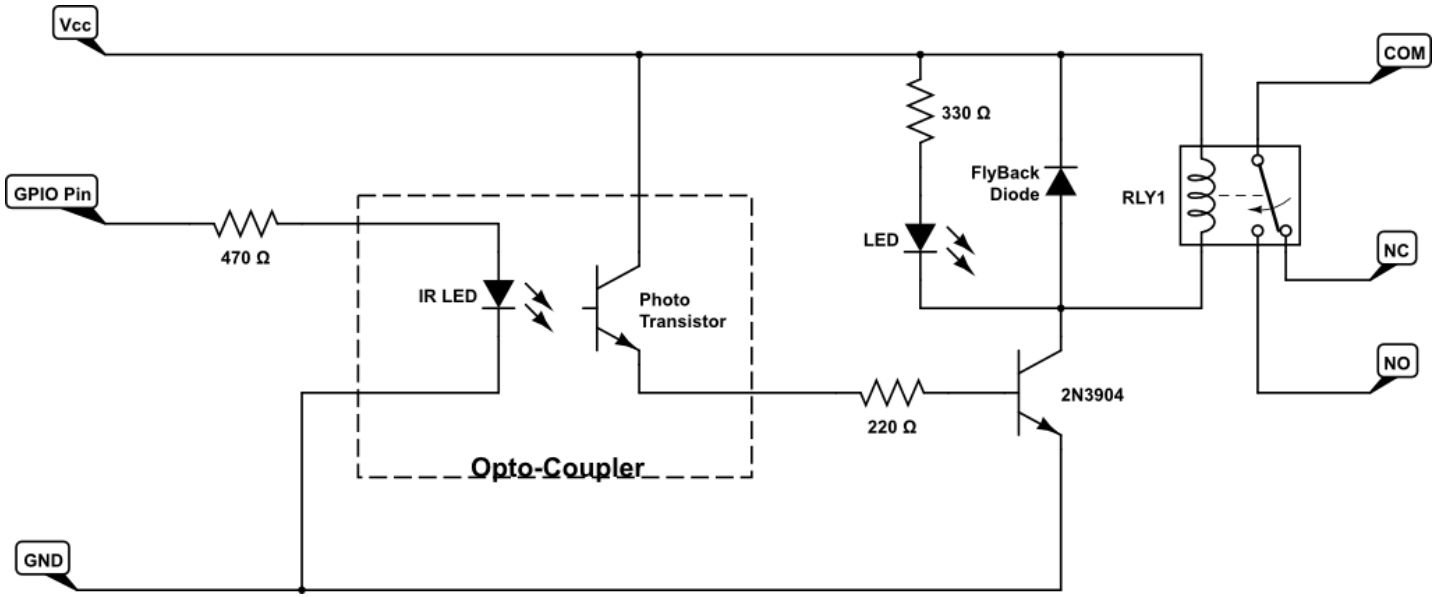
# Relays

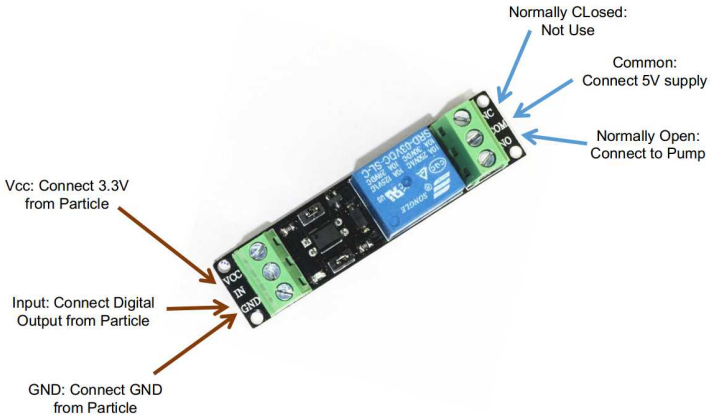



- When a device (e.g. a pump) requires higher voltage ($> 5V$) or higher current, then a relay can be used as a switch for the device
- The relay is activated by a digital pin from the microcontroller.
  - As the relay can require as much as 100mA from the digital pin, to provide sufficient current, use a current amplifying emitter follower to draw current directly from the USB connection ($V_{BUS}$).

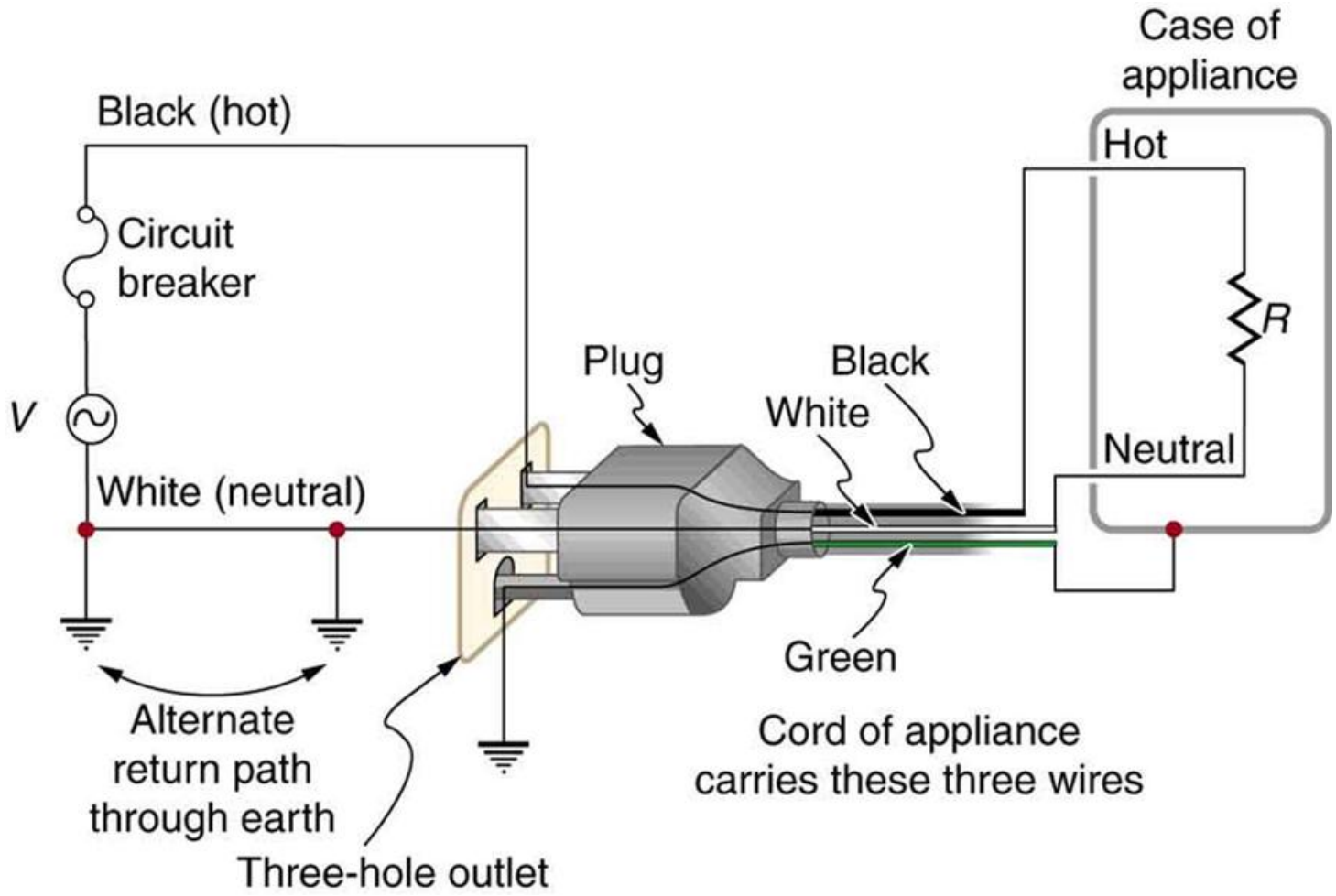
# Optocoupled Relay



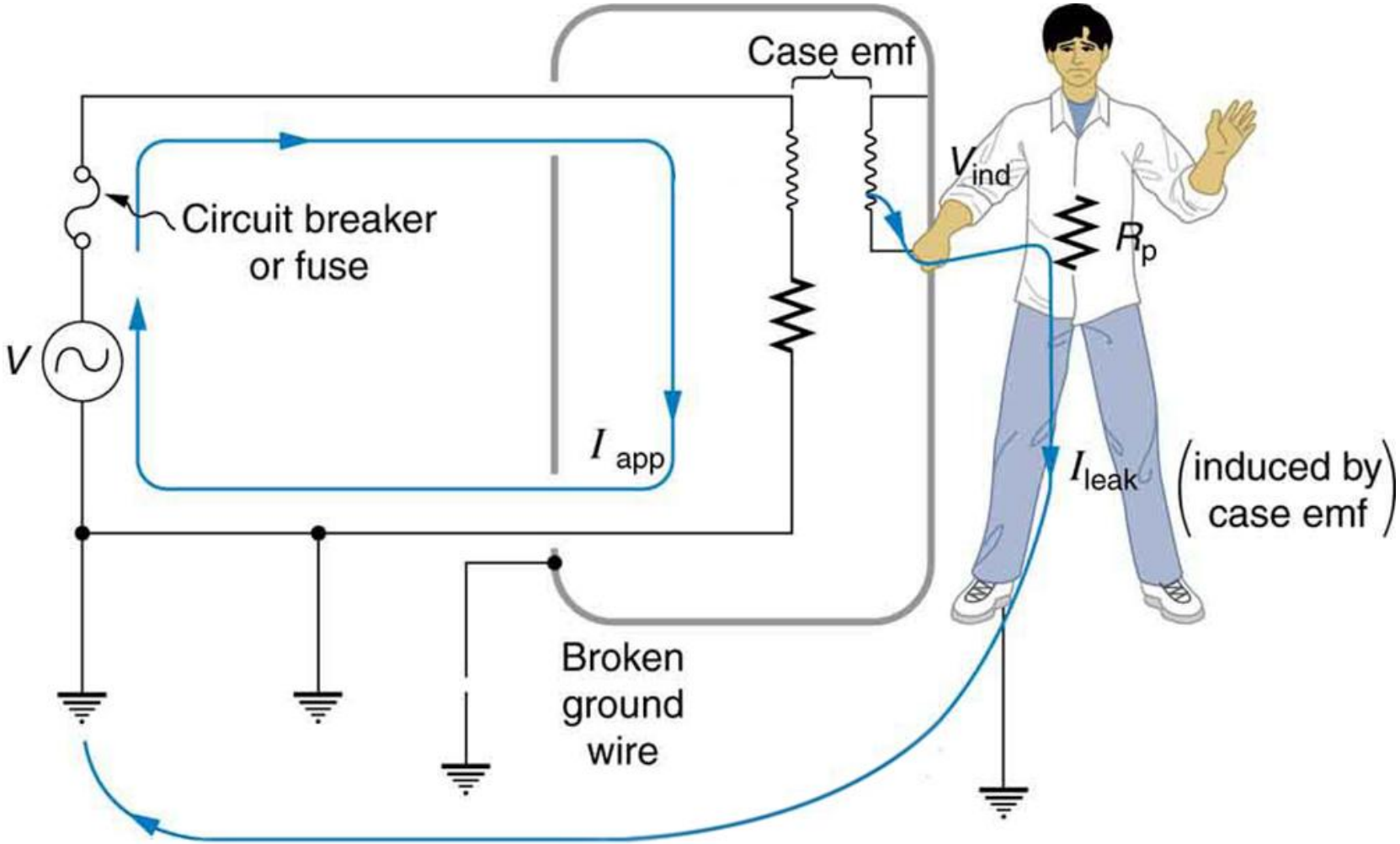




- Optocoupler isolates the relay load (which could be up to 240V) from the microcontroller electronics.

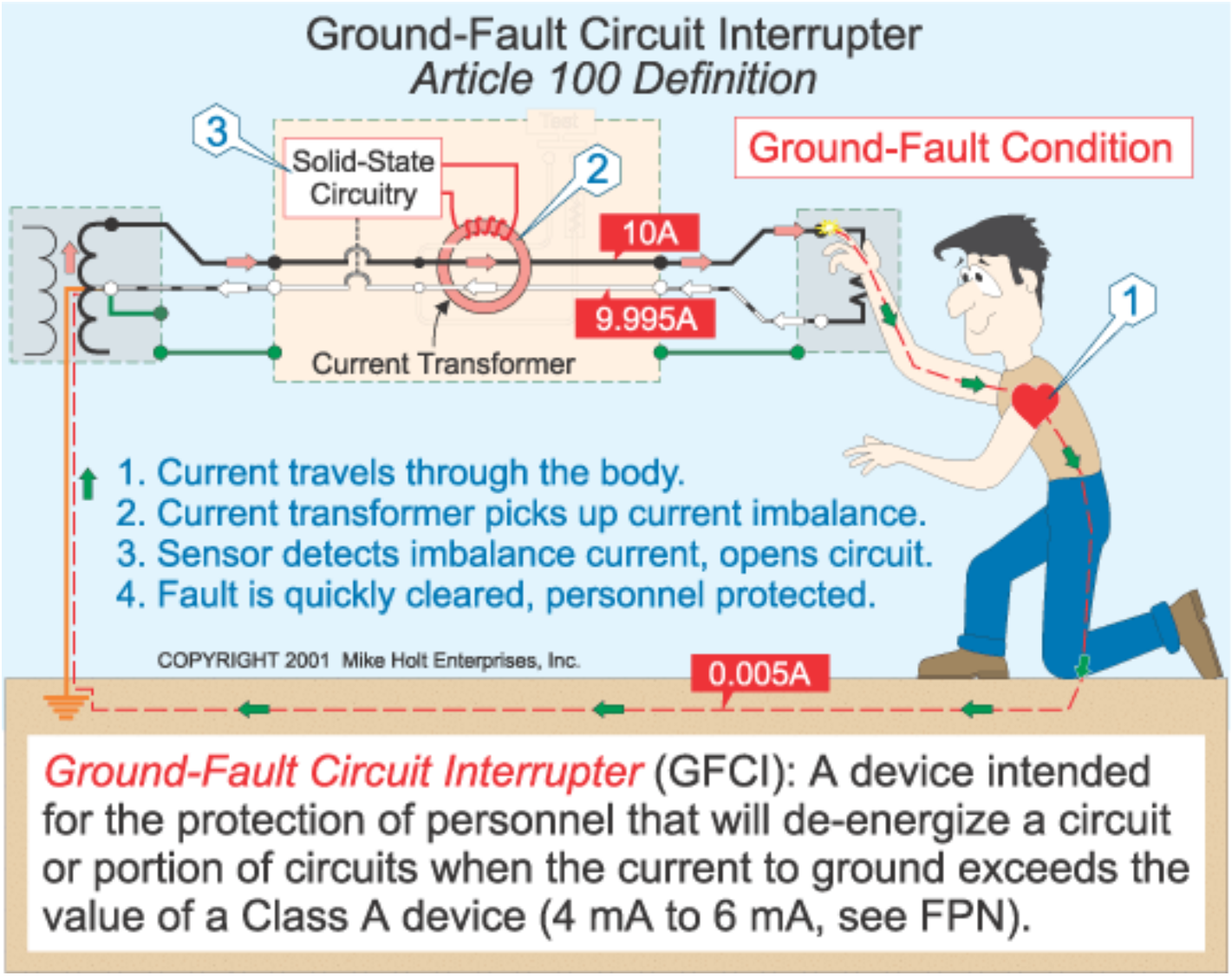
# Electrical Safety - Three Wires



# Electrical Safety - Electric Shock

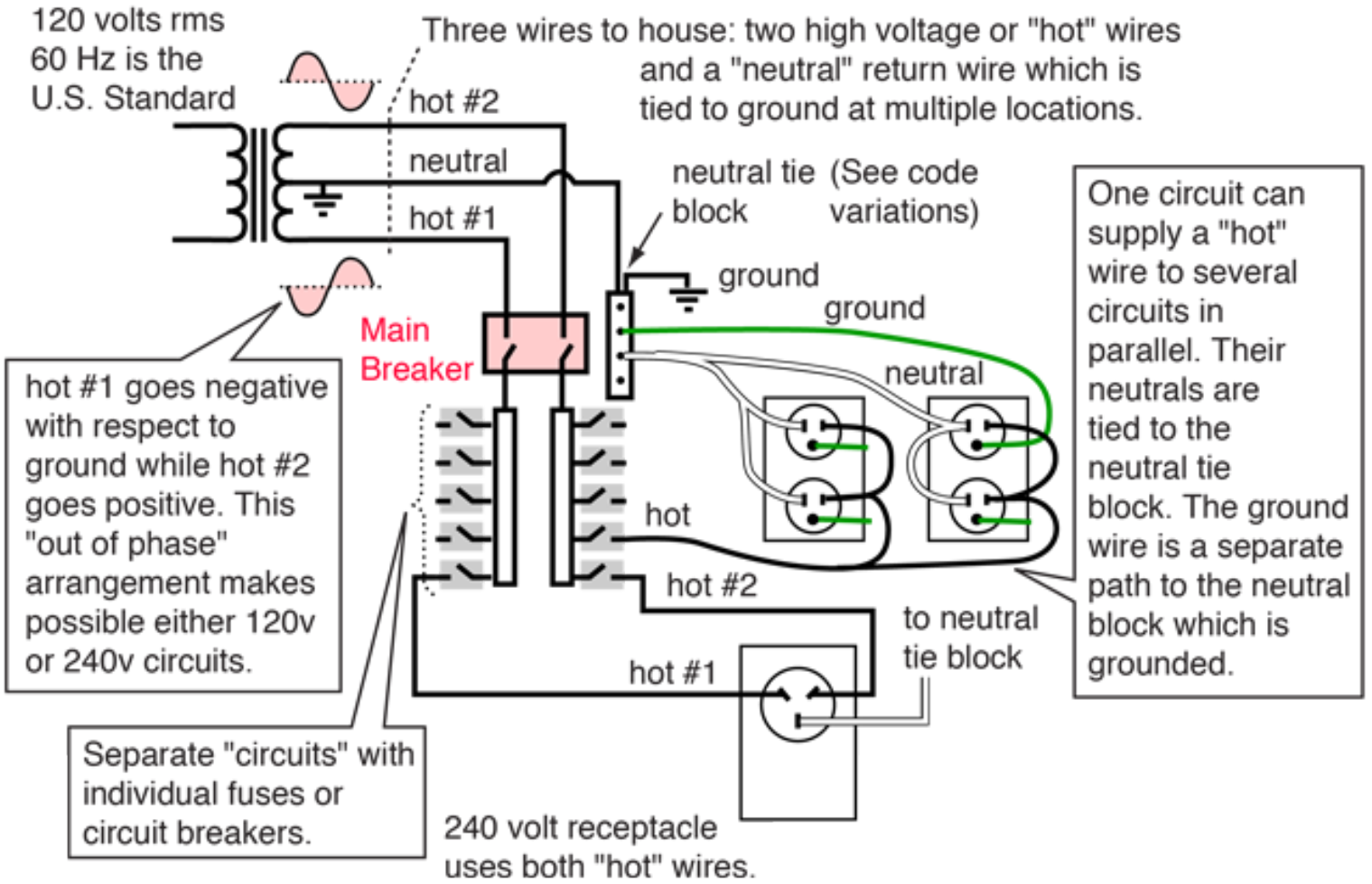


# Electrical Safety - GFCI



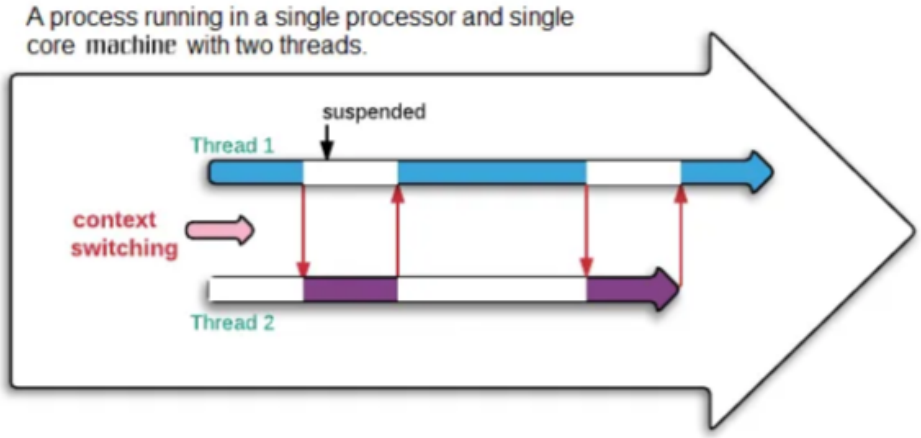

*Ground-Fault Circuit Interrupter* (GFCI): A device intended for the protection of personnel that will de-energize a circuit or portion of circuits when the current to ground exceeds the value of a Class A device (4 mA to 6 mA, see FPN).

# Electrical Safety - 240V



# Threads

Threads allow concurrent execution of multiple bits of code.

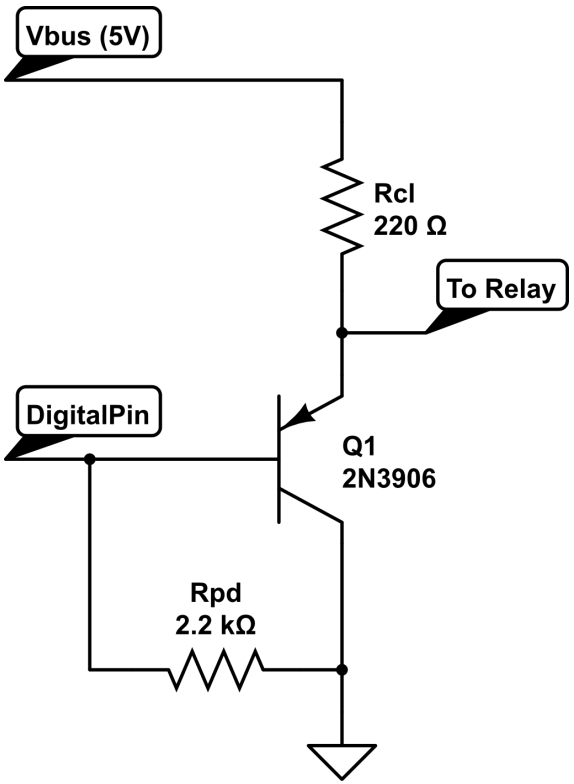



Threads are an advanced programming feature and very powerful when used correctly. Used incorrectly they can introduce new and novel issues into your code that are often more difficult to debug than single-threaded code

# Threaded Dust Sensor

```
1  float concentration;      // Global varialbe to store concentration
2  void getConc();           // Declare thread function
3  SYSTEM_THREAD(ENABLED);   // Enable threading
4
5  void setup() {
6      Serial.begin(9600);
7      new Thread("concThread", getConc); // Initiate thread
8  }
9  void loop() {
10     if ((millis()-lastTime) > updateTime) {
11       Serial.printf("Time: %0.2f, CONC: %0.2f\n",millis()/1000.0,concentration);
12       lastTime = millis();
13     }
14 }
15 void getConc() {                          // The thread
16   const int sampleTime = 30000;
17   unsigned int duration, startTime;
18   startTime = 0;
19   lowpulseoccupancy=0;
20   while(true) {                          // Run the below loop forever
21     duration = pulseIn(DUSTPIN, LOW);
22     lowpulseoccupancy = lowpulseoccupancy+duration;
23     if ((millis()-startTime) > sampleTime) {
24       ratio = lowpulseoccupancy/(sampleTime*10.0);
25       concentration = 1.1*pow(ratio,3)-3.8*pow(ratio,2)+520*ratio+0.62;
26       startTime = millis();
27       lowpulseoccupancy=0;
28     }
29   }
30 }
```

# Smart Houseplant Watering System

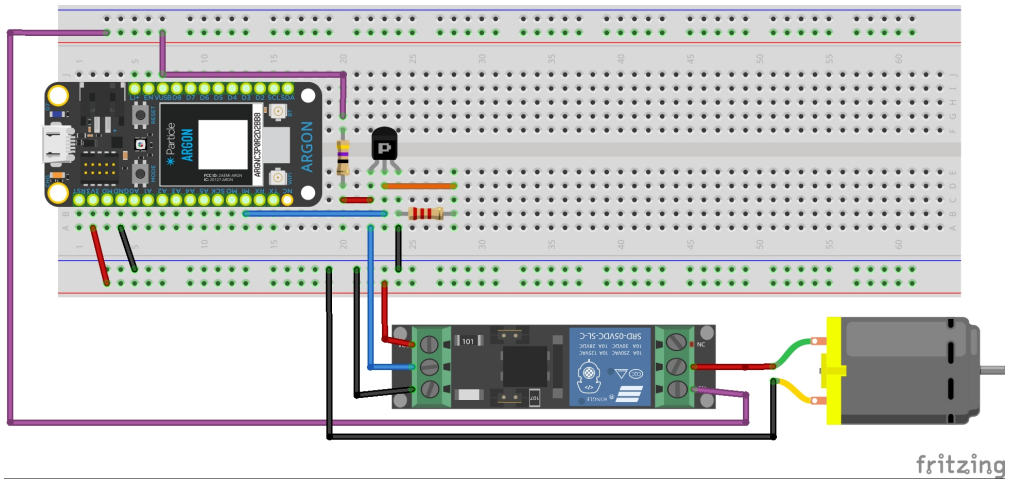



2N3906 Emitter Follower

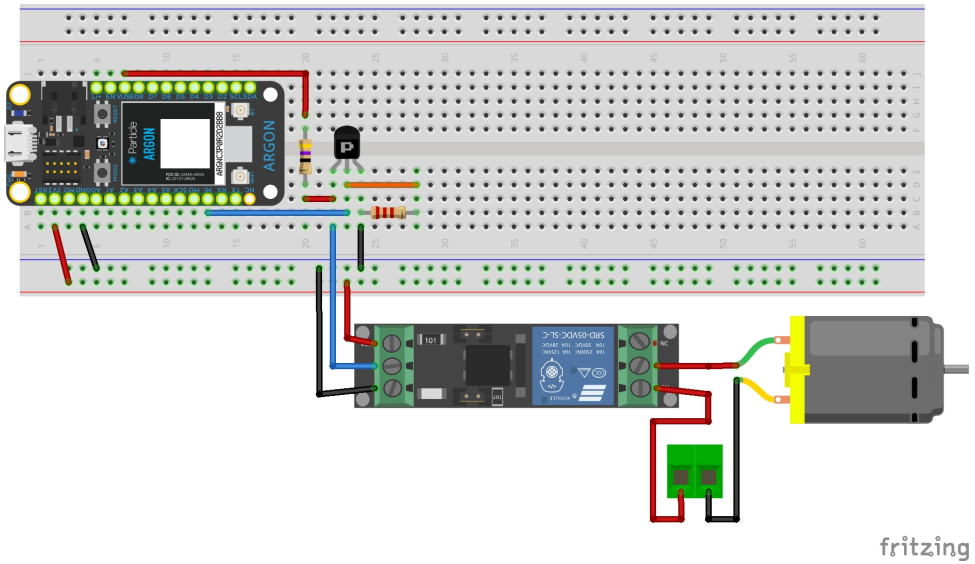
## Design

1. Components:
   - 2N3906 Emitter Follower and Relay
   - BME280 and SEEED sensors
   - OLED Display
2. Publish soil moisture and room environmental data to a new dashboard.
3. Automatically water your plant when the soil is too dry.
   - Only turn on the pump for a very short period of time ($\frac{1}{2} sec$).
4. Integrate a button into your dashboard that manually waters the plant.

# Relay and Pump Fritzing Diagram




If $V_{BUS}$ doesn't provide enough power, add external supply.



# Online Integrations Using Zapier

Zapier is an online automation tool that connects your favorite apps, such as Outlook, Slack, Mailchimp, and more. You can connect two or more apps to automate repetitive tasks.

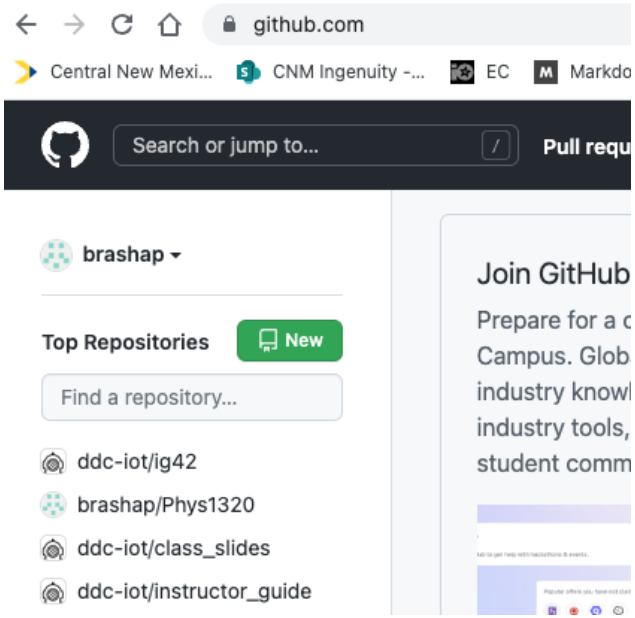
# Zapier Example


```
Adafruit_MQTT_Subscribe *subscription;
while ((subscription = mqtt.readSubscription(10000))) {
  if (subscription == &gotmail) {
    Serial.printf("You Got Mail \n");
    digitalWrite(D7, HIGH);
    flagServo.write(90);
    delay(10000);
    digitalWrite(D7, LOW);
    flagServo.write(5);
  }
}
```

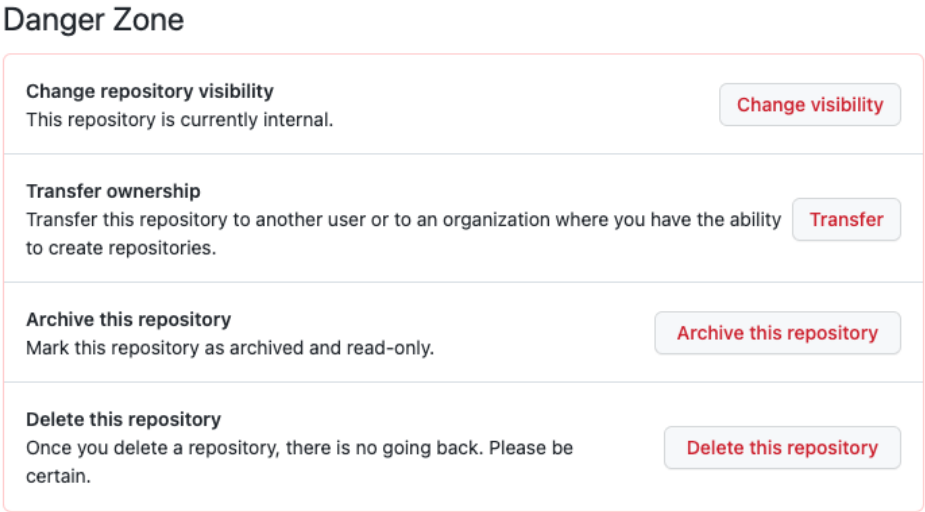
1. Create an Zapier account using your cnm.edu email and sign up for the 14-day "Professional" trial.
2. There are two Zapier examples in class_materials/ReferenceDocs:
   - Adafruit feed to email
   - Incoming email indicator to your Photon2
3. Add to your Midterm project to send yourself an email whenever your soil is too dry.
   In Zapier:
   - Create Feed Trigger from Adafruit
   - Add an Zapier filter
   - Add a Send to Email

# Midterm 2 - Github

Create a new repository




Under Security, change visibility to Public




IMPORTANT: Do not forget to add in the .gitignore file

- Must include credentials.h and \target
- Copy from any of the lessons

# Adding video to hackster.io story

## How do I add a video to my project?

Written by Benjamin Larralde
Updated over 2 years ago

To add a video, please upload it to Youtube or Vimeo first. You can leave it as unlisted if you prefer for it not to be accessible on those platforms.

Once the video is uploaded and you have a link for it, paste it in the story editor and press enter to embed it. Alternatively you can use the video button from the story editor toolbar:


Just make sure the cursor is placed where you want the video to be added first!

# Midterm 2 Expectations

1. Complete the Project Plan template from Brightspace and review with Instructors.
2. Integrate the entire system
   - Create a user friendly Adafruit.io dashboard
   - Buy and/or create a structure to hold the plant (in your IoT flowerpot), pump, and sensors.
   - Integration of SMS/email messaging using Zapier.
   - Video demo your Plant Watering System
3. Add to your portfolio
   - Add the project to your Hackster.io feed.
   - Create a Github repository (with descriptive README.md file) to use throughout this project.
   - Don't forget the .gitignore.
4. Class presentation - hackster, video demo, dashboard

# Module 12 - Memory: Bit, Bytes, and More

# Setting Up Second Photon 2

In order for you to take home your intact plant watering system, it is time to setup our second Photon 2:

1. Connect to WiFi
   - From VSCode: run L09_00_HelloReset
   - Or, use `https://docs.particle.io/tools/developer-tools/configure-wi-fi/`
2. Get your device's serial number using

```
particle serial identify

Your device id is e00fce681ffffffffc08949b
Your system firmware version is 5.8.2
```

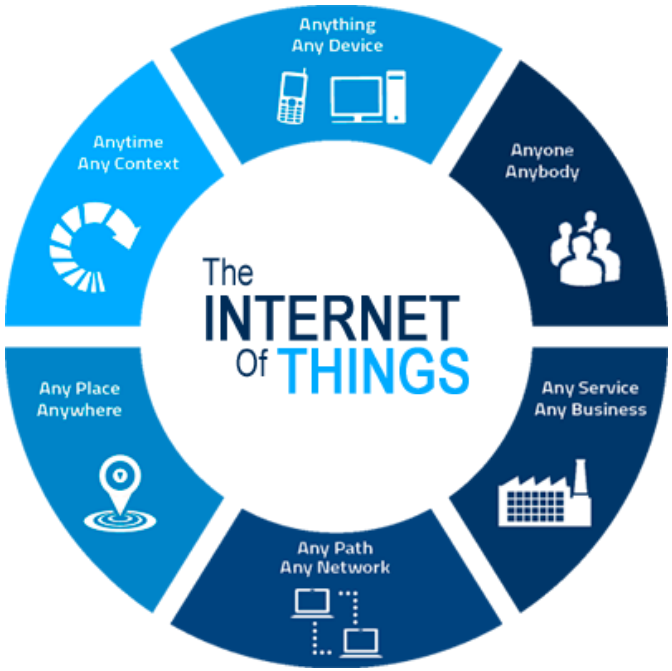
3. Claim the device to your account. Then, rename it to the name of your choice. This can only be done if it's breathing cyan.

```
particle device add e00fce681ffffffffc08949b
particle device rename e00fce681ffffffffc08949b deviceName
```

NOTE:

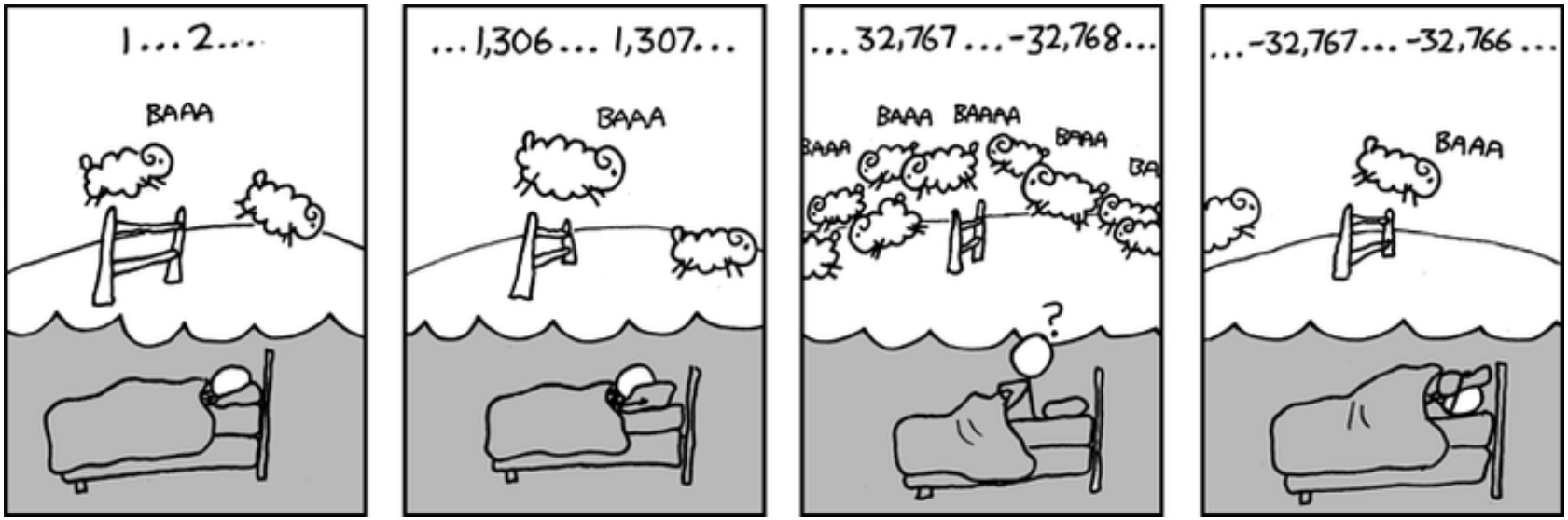
- Replace e00fce681ffffffffc08949b with the serial number of your device
- Replace deviceName with a name of your choosing

# Module 12 Objectives




- Learning Objectives
  1. Negative Numbers
  2. Bitwise Operations
  3. EEPROM
  4. Pointers
- Additional Items
  1. 3D Modeling Lesson 6 - Pin Wheels and Water Wheels
  2. Quiz 8

# Counting Sheep



# Negative Numbers

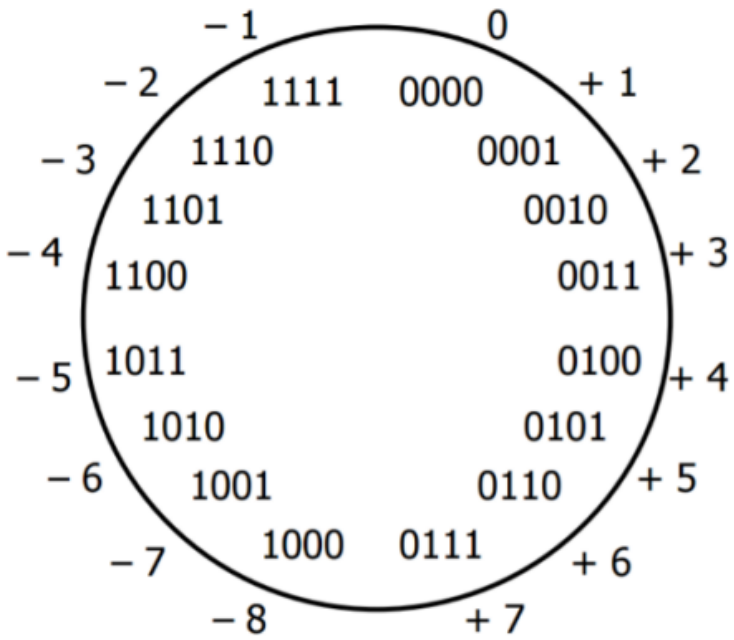
Question: How are negative numbers represented in binary?

| Type | Storage size | Value range |
|---|---|---|
| char | 1 byte | -128 to 127 or 0 to 255 |
| unsigned char | 1 byte | 0 to 255 |
| signed char | 1 byte | -128 to 127 |
| int | 2 or 4 bytes | -32,768 to 32,767 or -2,147,483,648 to 2,147,483,647 |
| unsigned int | 2 or 4 bytes | 0 to 65,535 or 0 to 4,294,967,295 |
| short | 2 bytes | -32,768 to 32,767 |
| unsigned short | 2 bytes | 0 to 65,535 |
| long | 4 bytes | -2,147,483,648 to 2,147,483,647 |
| unsigned long | 4 bytes | 0 to 4,294,967,295 |

Answer: Left-most bit is 1 to signify negative. But wait...

# 2's Compliment

2's compliment is used as it makes the math consistent.




| Integer | | 2's Complement |
|---|---|---|
| Signed | Unsigned | |
| 5 | 5 | 0000 0101 |
| 4 | 4 | 0000 0100 |
| 3 | 3 | 0000 0011 |
| 2 | 2 | 0000 0010 |
| 1 | 1 | 0000 0001 |
| 0 | 0 | 0000 0000 |
| -1 | 255 | 1111 1111 |
| -2 | 254 | 1111 1110 |
| -3 | 253 | 1111 1101 |
| -4 | 252 | 1111 1100 |
| -5 | 251 | 1111 1011 |

The negative plus the positive equals zero.

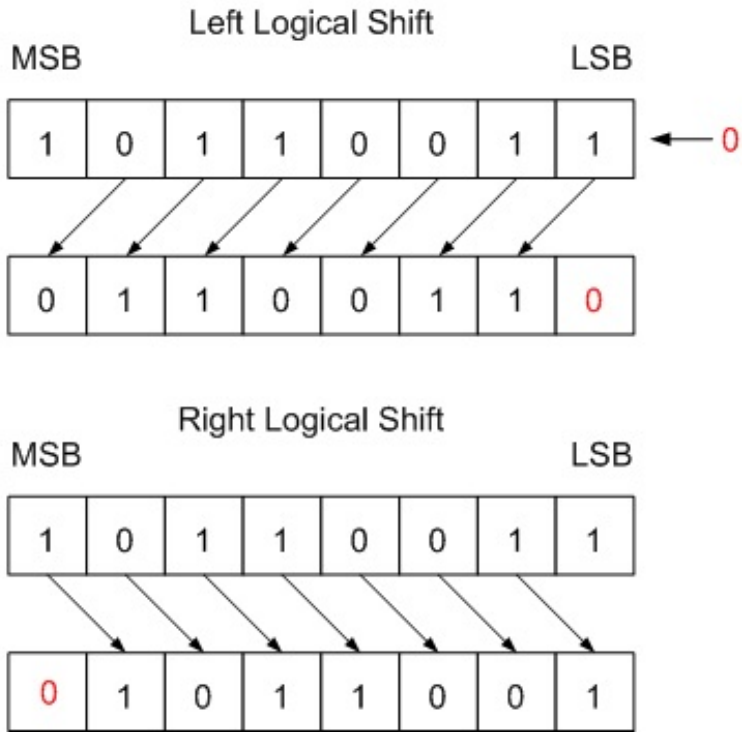
# Bitwise Operations

The following table lists the Bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then –

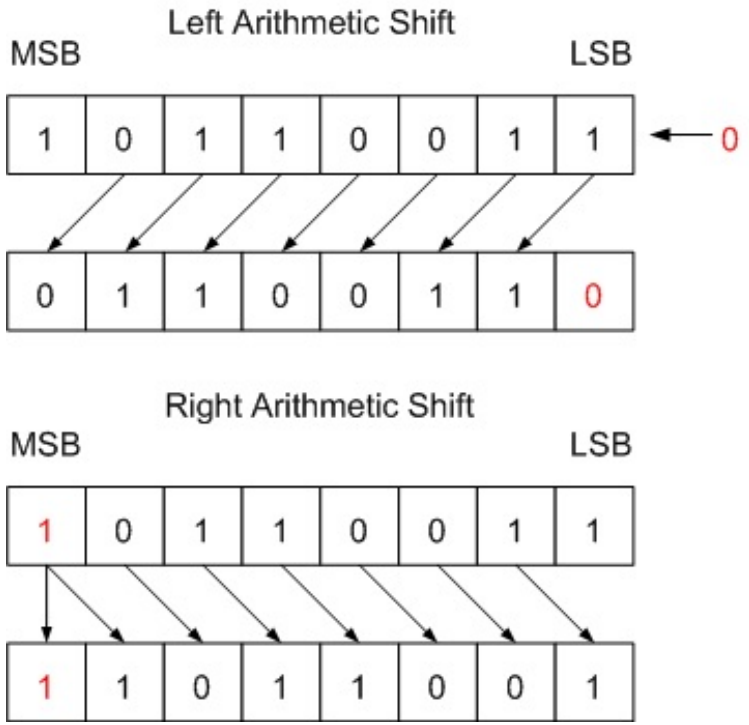
| Operator | Description | Example |
|---|---|---|
| & | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) = 12, i.e., 0000 1100 |
| \| | Binary OR Operator copies a bit if it exists in either operand. | (A \| B) = 61, i.e., 0011 1101 |
| ^ | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) = 49, i.e., 0011 0001 |
| ~ | Binary One's Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) = ~(60), i.e,. 1100 0011 |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 = 240 i.e., 1111 0000 |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 = 15 i.e., 0000 1111 |

# Bit Shifting

## Logical Bit Shift (Unsigned Int)




## Arithmetic Bit Shift (Int)




Whether the logical or arithmetic right shift is used depends on the datatype of the variable (unsigned or signed).

# Combining Bit-wise Operations

Find value of the $3^{rd}$ bit of 1110

- Right shift 1110 over 2 bits

$$1110 > 2 = 0011$$

- Select the rightmost bit with AND

$$0011 \ \& \ 0001 = 0001$$

- Putting it all together

$$(1110 > 2) \ \& \ 0001 = 0001$$

Replace $3^{rd}$ bit of 1001 with a 1

- Shift a 1 two bits to the left

$$0001 < 2 = 0100$$

- Replace the existing $3^{rd}$ with OR

$$1001|0100 = 1101$$

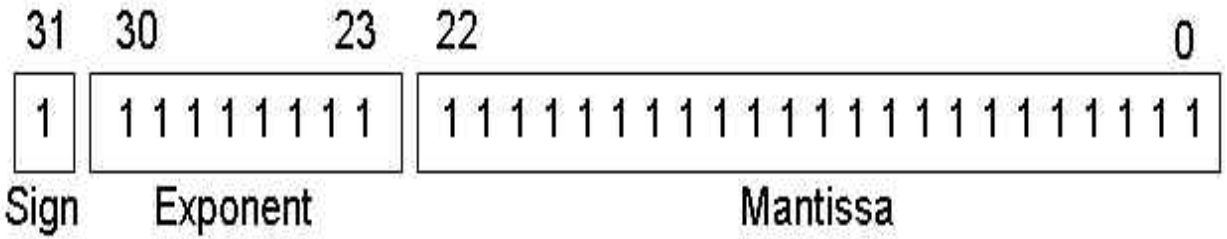
- Putting it all together

$$(0001 < 2)|1001 = 1101$$

How is this applied to bytes, both in binary and then hex?

# BONUS: But what about Floating Point

IEEE-754 Floating Point



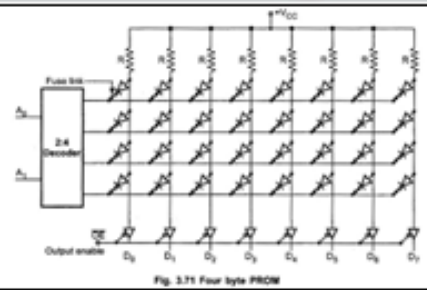

Example: In scientific notation: $-36382.36 = -3.638236 * 10^4$
With binary exponential equals $-1 * 1.1103014945983887 * 2^{15}$

- Sign: Negative = 1
- Exponent: 15 = 10001110 (with an exponent bias of 10000000)
- Mantissa: 1110301494598387 = 00011100001111001011100

Floating point representation is: 11000111000011100001111001011100

In printf() if %i is used instead of a %f: $-955,376,036$

# Types of Non-Volatile Memory

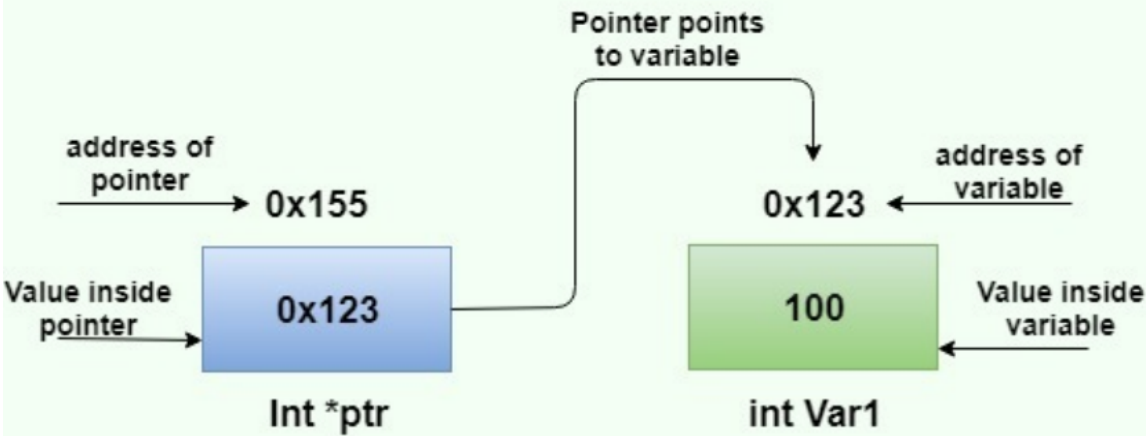
| | | |
|---|---|---|
| **ROM** | **Read Only Memory** - the data/code is defined by the circuts created during manufacturing. Used when the code/data never needs to be changed. |  |
| **PROM** | **Programmable Read Only Memory** - an array of fuses that can be "blown" to embed code/data. Program once, read many. |  |
| **EPROM** | **Erasable Read Only Memory** - programmable array that can be erased by exposing to UV light and then reprogrammed. The window in the packaging allows the UV to reach the EPROM circuits. Chip needs to be removed from device and placed in special UV oven to erase and then placed on the programmer. |  |
| **EEPROM** | **Electrically Erasable Read Only Memory** - an EPROM that can be electrically erased and reprogrammed without being removed. Computer BIOS (basic input output system), cell phone memory, USB stick |  |

# Electrically Erasable Programmable Read Only Memory

EEPROM emulation allows small amounts of data to be stored and persisted even across reset, power down, and user and system firmware flash operations. Since the data is spread across a large number of flash sectors, flash erase-write cycle limits should not be an issue in general.

```
len = EEPROM.length(); //available EEPROM bytes
// Photon2s have 4096 bytes of emulated EEPROM.
// Addresses 0x0000 through 0x0FFF

addr = 0x00AE;    //addr between 0 and len-1

val = 0x45;
EEPROM.write(addr, val);

value = EEPROM.read(addr);
```

# Pointers



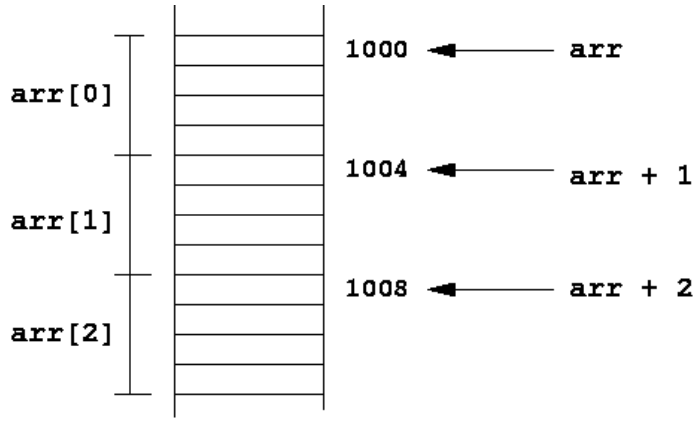

1. A pointer is a variable whose value is the address of another variable.
2. When you declare a pointer, the $*$ symbol denotes that this variable is a pointer variable. For example:
   - Pointer to an Integer: int *ptr;
3. Reference operator (&) gives the address of a variable.
4. To get the value stored in the memory address, we use the dereference operator ($*$).

# Pointers

```
1  int data = 13;
2  int data2;
3  int *ptr;
4
5  void setup() {
6    Serial.begin(9600);
7    delay(1000);
8    ptr = &data;           //point ptr to the memory location of data
9    data2 = *ptr;          //set data2 to value of data (13)
10
11   // Print the Value and Address of the Variables
12   Serial.printf("Variable          Value       Address \n");
13   Serial.printf("  data             %i        0x%X    \n",data, &data);
14   Serial.printf("  ptr            0x%X        0x%X    \n",ptr, &ptr);
15   Serial.printf("  data2            %i        0x%X    \n",data2,&data2);
16 }
```

```
Serial monitor opened successfully:
Variable          Value       Address
  data              13        0x2003E380
  ptr          0x2003E380     0x2003E3F4
  data2             13        0x2003E3F0
```

# Pointers and Arrays




```
1 int arr[] = {100, 200, 300};
2
3 void loop() {
4   // Compiler converts below to *(arr + 2).
5   Serial.printf("%i \n", arr[2]);
6
7   // So below also works.
8   Serial.printf("%i \n", *(arr + 2));
9 }
```

When an array (arr[]) is declared, the variable is a pointer to the first element of a continuous block of memory. In this case there are 3 elements, each 4-bytes in size, for a total of 12-bytes.

# Finding Average of an Array

```
// This function finds the average of an array.
// The array is passed to it as a pointer.

float getAverage(int *array,int size) {
  int j;
  float total=0;
  for(j=0;j<size;j++) {
    total += array[j];
  }
  return total/size;
}
```

# Finding Average of Arrays in Action

```
1  int xArray[4], yArray[256];
2  int *pointerX, *pointerY;
3  float average;
4  int i, sizeX, sizeY;
5
6  void setup() {
7    pointerX=&xArray[0];
8    sizeX=sizeof(xArray)/4;
9    pointerY=&yArray[0];
10   sizeY=sizeof(yArray)/4;
11   for(i=0;i<sizeX;i++) {
12     xArray[i] = random(0,255);
13   }
14   for(i=0;i<sizeY;i++) {
15     yArray[i] = random(256,512);
16   }
17   average = getAverage(pointerX, sizeX);
18   average = getAverage(pointerY, sizeY);
19 }
```

```
Array X Average = 162.50
Array Y Average = 388.35
xArray[0] value: 173, *pointerX:     173, pointerX:   0x2003E3DC
xArray[1] value: 179, *(pointerX+1): 179, pointerX+1: 0x2003E3E0
xArray[2] value: 110, *(pointerX+2): 110, pointerX+2: 0x2003E3E4
xArray[3] value: 188, *(pointerX+3): 188, pointerX+3: 0x2003E3E8
```

# Returning Multiple Values from a Function

Arguments can be passed to a function by reference; thus, allowing multiple parameters to be returned by a function.

```
1  const int BILLBUTTON = D4;
2  float table1, table2;  // subtotal for each table
3  float tax1, tax2, tip1, tip2;
4  bool billReady;
5
6  void calcBill(float subtotal, float *tax, float *suggestedTip);
7
8  void setup() {
9    // NOT SHOWN: Setup of Serial Monitor, pinMode, etc.
10   table1 = 23.76;
11   table2 = 47.29;
12 }
13
14 void loop() {
15   billReady = digitalRead(BILLBUTTON);
16   if(billReady) {
17     calcBill(table1,&tax1,&tip1);// pass by reference, memory address sent to function
18     calcBill(table2,&tax2,&tip2);// the function uses pointers to place values in memory
19     Serial.printf("Table1\nSubtotal:%0.2f\nTax:%0.2f\nSuggested Tip:%0.2f\n",table1,tax1,
         tip1);
20   }
21 }
22
23 void calcBill(float subtotal, float *tax, float *suggestedTip) {
24   *tax = subtotal * 0.07;
25   *suggestedTip = subtotal * 0.18;
26 }
```

# Variables and Pointers - Values and Addresses

```
1  int x,y,sum,diff;
2  void addSubtract(int firstNum, int secondNum, int *theSum, int *theDiff);
3
4  void setup() {
5    x = 7;
6    y = 2;
7    addSubtract(x,y,&sum,&diff);
8  }
9
10 void loop() {}
11
12 void addSubtract(int firstNum, int secondNum, int *theSum, int *theDiff) {
13
14   *theSum = firstNum + secondNum;
15   *theDiff = firstNum - secondNum;
16 }
```

```
Before Function Call
x    = 7, &x    = 1007AE9C
y    = 2, &y    = 1007AEA0
sum =  0, &sum  = 1007AE98
diff = 0, &diff = 1007AE94
```

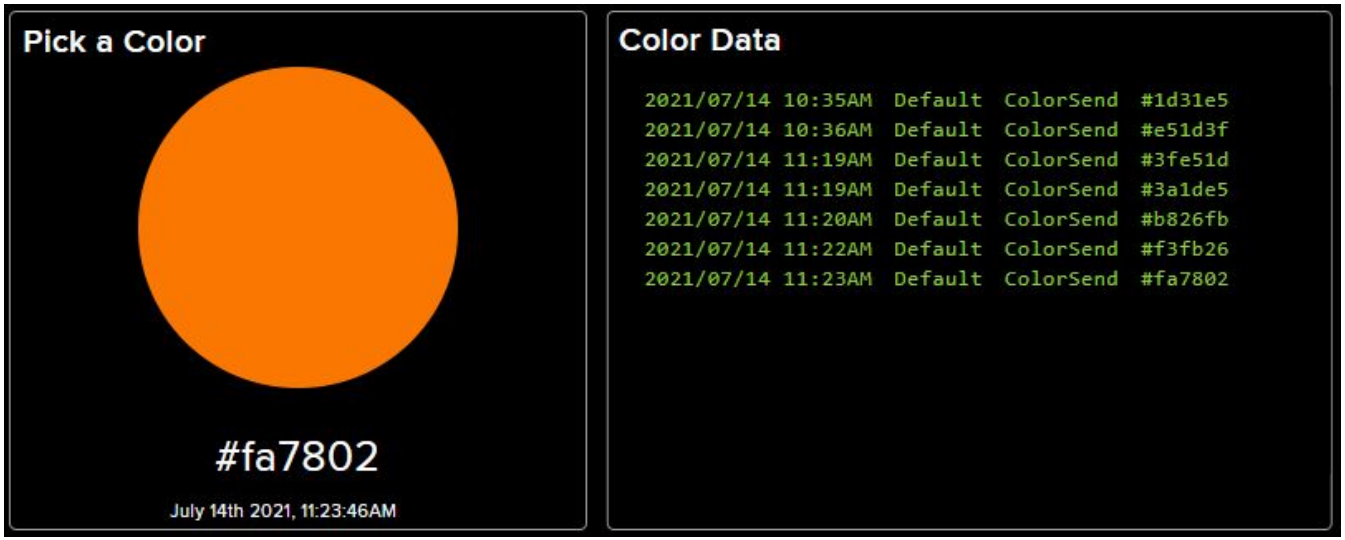
```
At end of function
firstNum  = 7,     &firstNum  = 10011DDC
secondNum = 2,     &secondNum = 10011DD8
*theSum  = 1007AE98, &theSum  = 10011DD4
*theDiff = 1007AE94, &theDiff = 10011DD0
```

```
After Function Call
x    = 7, &x    = 1007AE9C
y    = 2, &y    = 1007AEA0
sum  = 9, &sum  = 1007AE98
diff = 5, &diff = 1007AE94
```

# memcpy(), (char *), and strtol()

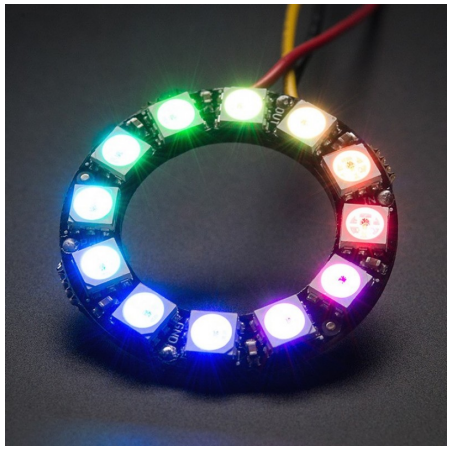
```
int color;
byte data[] = {0x23,0x42,0x41,0x34,0x32,0x35,0x44,0x39,0x35};
byte buf[6];


/* memcpy() - copy from specific memory locations to new locations
 * memcpy(to, from, size);
 *     to -> pointer to starting address of where to copy to
 *     from -> pointer to starting address of where to copy from
 *     size -> number of btyes to copy
 */

memcpy(buf, &data[1],6);     //copy bytes 1 through 6 and place in buf

/* (char *) - typecasting a data type to a char-type pointer */

Serial.printf("Converting the data array to ascii symbols returns %s,\n",(char *)data);


/* strtol() - string to long - similar to atoi()
 * strtol(charString, end, base)
 *     charString -> string that contains number to be converted
 *     end -> character to end conversion on (set to NULL)
 *     base -> base of integer (16 for hex)
 */

color = strtol((char *)buf,NULL,16); // convert string to int (hex)
```

# EXAMPLE: Adafruit MQTT Subscribe - Color Picker




```
int color;
byte buf[6];

  Adafruit_MQTT_Subscribe *subscription;
  while ((subscription = mqtt.readSubscription(1000))) {
    if (subscription == &mqttColor) {
      Serial.printf("Received from Adafruit: %s \n",(char *)mqttColor.lastread);
      memcpy(buf, &mqttColor.lastread[1],6);      //strip off the '#'
      Serial.printf("Buffer: %s \n",(char *)buf);
      color = strtol((char *)buf,NULL,16);        // convert string to int (hex)
      Serial.printf("Buffer: 0x%02X \n",color);
    }
  }
```

# L12_Memory Assignments



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L12_01_ColorPicker
   - Create a feed and dashboard on Adafruit.io using the Color Picker block.
   - Subscribe to your Color Picker feed and convert the lastRead() to a integer (hex)
   - Light up your NeoPixel ring the received color.
   - Using pointers create a function to convert the hex color into individual R,G,B components using Bit Shifting and AND.
   - From void loop, store the components of the color as bytes in the Particle's EEPROM.
2. L12_02_RetrieveShow
   - Set pixel ring to white for at least 1 second
   - Retrieve the color from EEPROM memory.
   - Convert to a hex color code (e.g. 0xABCDEF)
   - Display the color on the NeoPixel ring using setPixelColor(n,hexColor)

# Useful Properties: Identity Element

An identity element is a special type of element of a set with respect to a binary operation on that set, which leaves any element of the set unchanged when combined with it.

Addition:

$$x + 0 = x \tag{1}$$

Multiplication:

$$x * 1 = x \tag{2}$$

Bitwise AND:

$$x \& 1 = x \tag{3}$$

Bitwise OR:

$$x | 0 = x \tag{4}$$

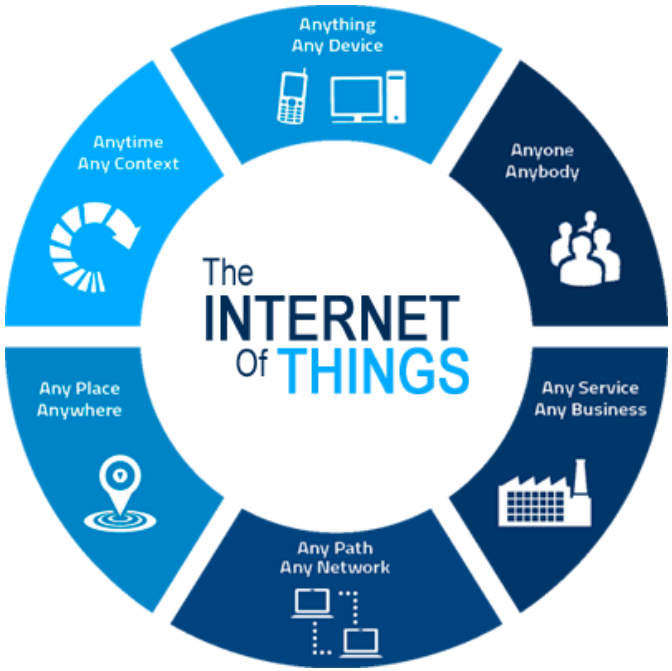
# Bonus: Passing Objects via Reference

```
1  // Declare Opjects
2  Adafruit_NeoPixel pixel(30, SPI1, WS2812B);
3
4  // Declare pixelFill function with passing in neopixel object by reference
5  void pixelFill(Adafruit_NeoPixel  *strip, int start, int end, int pixColor);
6
7  SYSTEM_MODE(SEMI_AUTOMATIC);
8
9  void setup() {
10   pixel.begin();
11 }
12
13 void loop() {
14   pixelFill(&pixel, 0, 30, 0x0000FF); // note the pass by reference "&pixel"
15 }
16
17 // Light up a segment of pixels
18 void pixelFill(Adafruit_NeoPixel *strip,int start, int end, int pixColor) {
19   int i;
20
21   for(i=start;i<=end;i++) {
22     strip->setPixelColor(i,pixColor); //note the "->" instead of the "."
23   }
24   strip->show();
25 }
```

# Added Bonus: Array of Functions via Pointers

```
1  //Array of pointers to each of the functions
2  int (* funky[4])(int x, int y) = {add,sub,mult,divi};
3
4  int a,b,answer,i;
5
6  void setup() {
7    Serial.begin(9600);
8  }
9
10 void loop() {
11   a = random(0,100);
12   b = random(0,100);
13   for(i=0;i<4;i++) {
14     answer = funky[i](a,b);
15     Serial.printf("For function %i: a = %i and b = %i equals %i \n",i,a,b,answer);
16     delay(250);
17   }
18   Serial.printf("\n\n\n");
19   delay(3000);
20 }
21
22 // The Functions
23 int add(int x,int y) {return x+y;}
24
25 int sub(int x,int y) {return x-y;}
26
27 int mult(int x,int y) (return x*y;}
28
29 int divi(int x,int y) {return x/y;}
```

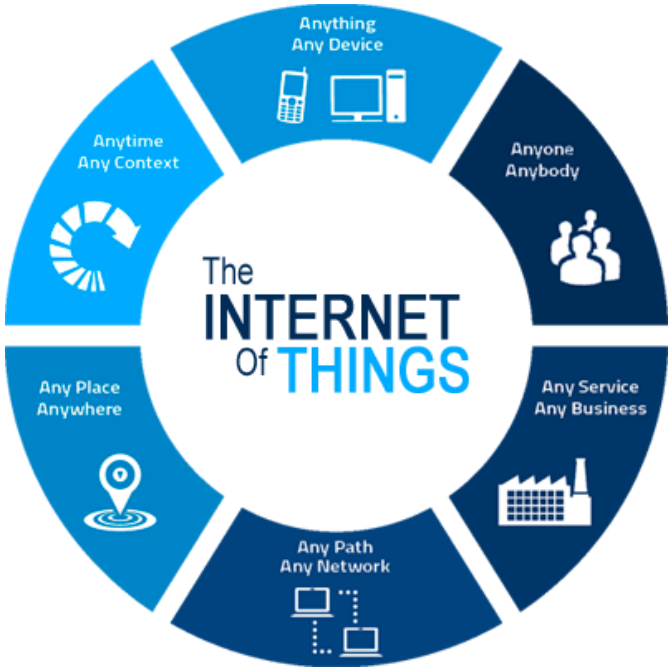
# Module 12 Review




- Learning Objectives
  1. Negative Numbers
  2. Bitwise Operations
  3. EEPROM
  4. Pointers
- Additional Items
  1. 3D Modeling Lesson 6 - Pin Wheels and Water Wheels
  2. Quiz 8

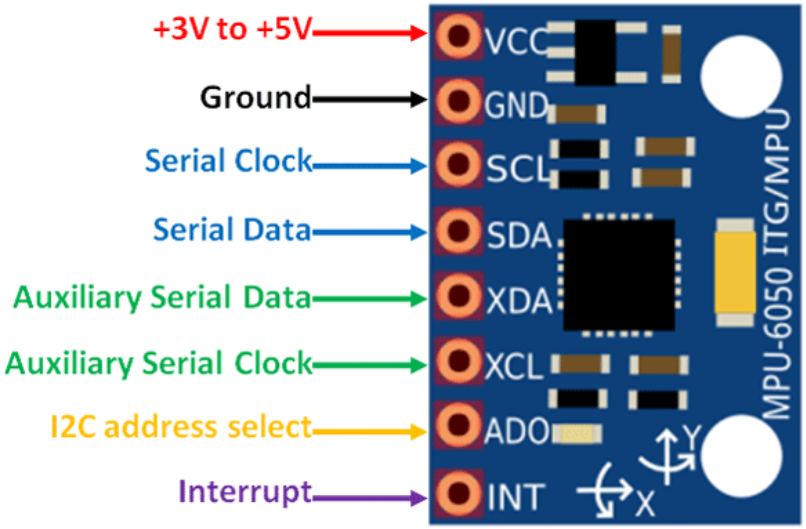
# Module 13 - Motion

# Module 13 Objectives




- Learning Objectives
  1. Utilized a Data Sheet to learn to access a component
  2. Accelerometers and Gyroscopes
  3. A little big of trig and calculus
  4. Stepper motors
  5. Hall effect sensors
  6. Interrupts
- Additional Items
  1. Quiz 9

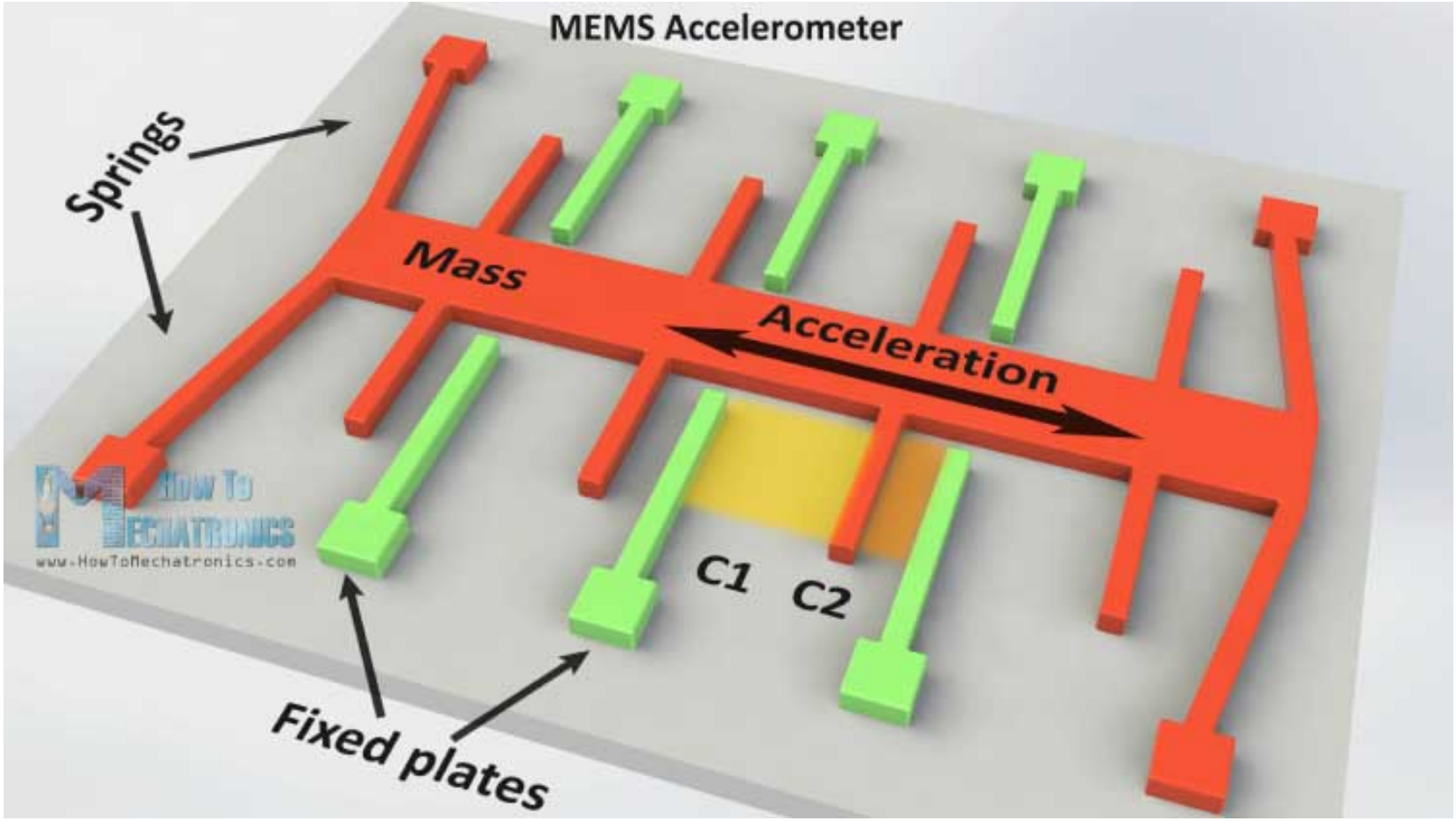
# MPU6050 Accelerometer



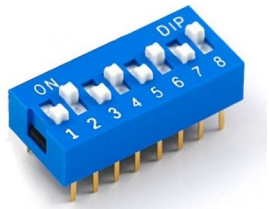

- Data Output: signed 16 Bit
- Accel range:±2 ±4 ±8 ±16g
- Gyros range:±250 500 1000 2000°/s

- XDA and XCL refer to the I2C bus that the MPU-6050 controls, so it can read from slave devices such as magnetometers etc.
- ADO pin changes I2C address when grounded.
- The interrupt pin notifies the MPU about available data. To reduce power consumption, the processor can go into sleep mode and the interrupt can be used to wake up the processor.

# Accelerometers



# DIP Switches and Register Maps




Remember: serial usb setup-done

Table 18: Memory map

| Register Name | Address | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 | Reset state |
|---|---|---|---|---|---|---|---|---|---|---|
| hum_lsb | 0xFE | hum_lsb<7:0> | | | | | | | | 0x00 |
| hum_msb | 0xFD | hum_msb<7:0> | | | | | | | | 0x80 |
| temp_xlsb | 0xFC | temp_xlsb<7:4> | | | | 0 | 0 | 0 | 0 | 0x00 |
| temp_lsb | 0xFB | temp_lsb<7:0> | | | | | | | | 0x00 |
| temp_msb | 0xFA | temp_msb<7:0> | | | | | | | | 0x80 |
| press_xlsb | 0xF9 | press_xlsb<7:4> | | | | 0 | 0 | 0 | 0 | 0x00 |
| press_lsb | 0xF8 | press_lsb<7:0> | | | | | | | | 0x00 |
| press_msb | 0xF7 | press_msb<7:0> | | | | | | | | 0x80 |
| config | 0xF5 | t_sb[2:0] | | | filter[2:0] | | | | spi3w_en[0] | 0x00 |
| ctrl_meas | 0xF4 | osrs_t[2:0] | | | osrs_p[2:0] | | | mode[1:0] | | 0x00 |
| status | 0xF3 | | | | | measuring[0] | | | im_update[0] | 0x00 |
| ctrl_hum | 0xF2 | | | | | | osrs_h[2:0] | | | 0x00 |
| calib26..calib41 | 0xE1…0xF0 | calibration data | | | | | | | | individual |
| reset | 0xE0 | reset[7:0] | | | | | | | | 0x00 |
| id | 0xD0 | chip_id[7:0] | | | | | | | | 0x60 |
| calib00..calib25 | 0x88…0xA1 | calibration data | | | | | | | | individual |

| Registers: | Reserved registers | Calibration data | Control registers | Data registers | Status registers | Chip ID | Reset |
|---|---|---|---|---|---|---|---|
| Type: | do not change | read only | read / write | read only | read only | read only | write only |

# REMINDER - Data Types: Numbers

| Data Type | 8-bit AVR systems (Arduino Uno) | | | 32-bit ARM systems (Teensy 3.2) | | |
|---|---|---|---|---|---|---|
| | bytes | range (signed) | range (unsigned) | bytes | range (signed) | range (unsigned) |
| char | 1 | -128 to 127 | 0 to 255 | 1 | -128 to 127 | 0 to 255 |
| short | 2 | +/- 32,767 | 0 to 65,353 | 2 | +/- 32,767 | 0 to 65,353 |
| int | 2 | +/- 32,767 | 0 to 65,353 | 4 | +/- 2,147,483,648 | 0 - 4,294,967,295 |
| long | 4 | +/- 2,147,483,648 | 0 - 4,294,967,295 | 4 | +/- 2,147,483,648 | 0 - 4,294,967,295 |
| long long | 8 | +/- 9,223,372,036,854,770,000 | 0 to 18,446,744,073,709,551,615 | 8 | +/- 9,223,372,036,854,770,000 | 0 to 18,446,744,073,709,551,615 |
| float | 4 | 3.4E +/- 38 (7 digits) | n/a | 4 | 3.4E +/- 38 (7 digits) | n/a |
| double | 4 | 3.4E +/- 38 (7 digits) | n/a | 8 | 1.7E +/- 308 (15 digits) | n/a |
| long double | 8 | 1.7E +/- 308 (15 digits) | n/a | 8 | 1.7E +/- 308 (15 digits) | n/a |
| Unambiguous | | | | | | |
| uint8_t | 1 | n/a | 0 to 255 | 1 | n/a | 0 to 255 |
| int8_t | 1 | -128 to 127 | n/a | 1 | -128 to 127 | n/a |
| uint16_t | 2 | n/a | 0 to 65,353 | 2 | n/a | 0 to 65,353 |
| int16_t | 2 | +/- 32,767 | n/a | 2 | +/- 32,767 | n/a |
| uint32_t | 4 | n/a | 0 - 4,294,967,295 | 4 | n/a | 0 - 4,294,967,295 |
| int32_t | 4 | +/- 2,147,483,648 | n/a | 4 | +/- 2,147,483,648 | n/a |

The MPU6050 provides acceleration as a signed 16-bit number. The data type int16_t should be used to ensure the sign-bit is in the correct location.

# Initializing MPU6050

```
// Initialize the MPU in the void setup()
void setup() {
  // Begin I2C communications
  Wire.begin();

  // Begin transmission to MPU-6050
  // MPU_ADDR should be declared as a CONST INT
  Wire.beginTransmission(MPU_ADDR);

  // Select and write to PWR_MGMT1 register
  Wire.write(0x6B);
  Wire.write(0x00); // wakes up MPU-6050

  // End transmission and close connection
  Wire.endTransmission(true);
}
```

# Reading Acceleration Data from the MPU-6050

```
// Declare variables
byte accel_x_h, accel_x_l;    //variables to store the individual btyes
int16_t accel_x;              //variable to store the x-acceleration

void loop() {
  // Set the "pointer" to the 0x3B memory location of the MPU and wait for data
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x3B); // starting with register 0x3B
  Wire.endTransmission(false); // send the set pointer command and keep active.

  // Request and then read 2 bytes
  // Syntax:
  //    Wire.requestFrom(I2C_addr, quantity, stop);
  //    Wire.read(); //repeat this for each byte to be read

  Wire.requestFrom(MPU_ADDR, 2, true);
  accel_x_h = Wire.read(); // x accel MSB
  accel_x_l = Wire.read(); // x accel LSB

  accel_x = accel_x_h << 8 | accel_x_l;     // what happens if declared int instead?
  Serial.printf("X-axis acceleration is %i \n",accel_x);
}
```

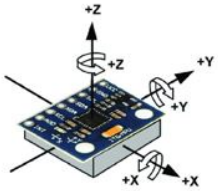
*Note: the data is stored in Big Endian Byte Order. The most significant byte (the "big end") of the data is placed at the byte with the lowest address. The rest of the data is placed in the next byte.*

# Tale of Three Hexes

Hex values are used three different ways in our $I^2C$ communications:

1. $I^2C$ Address
   - Wire.beginTransmission()
   - Wire.endTransmission()
   - Wire.requestFrom()
2. Set a pointer to a register (memory location)
   - The first Wire.write() after Wire.beginTransmission
3. As Data
   - All subsequent Wire.write() after the first
   - Wire.read()

# Assignment: L13_Motion




- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L13_01_MPU6050
   - Read values from the registers associated with X, Y, and Z acceleration.
   - Convert the returned acceleration values to standard gravity units (e.g. when flat on the table, $a_z$ = -1G).
2. L13_02_AutoRotate
   - Display date and time on an OLED display.
   - Use accel values to auto-rotate the OLED.
3. Extra
   - Modify L13_01_MPU6050 to be able to modify range/sensitivity with a button or encoder.

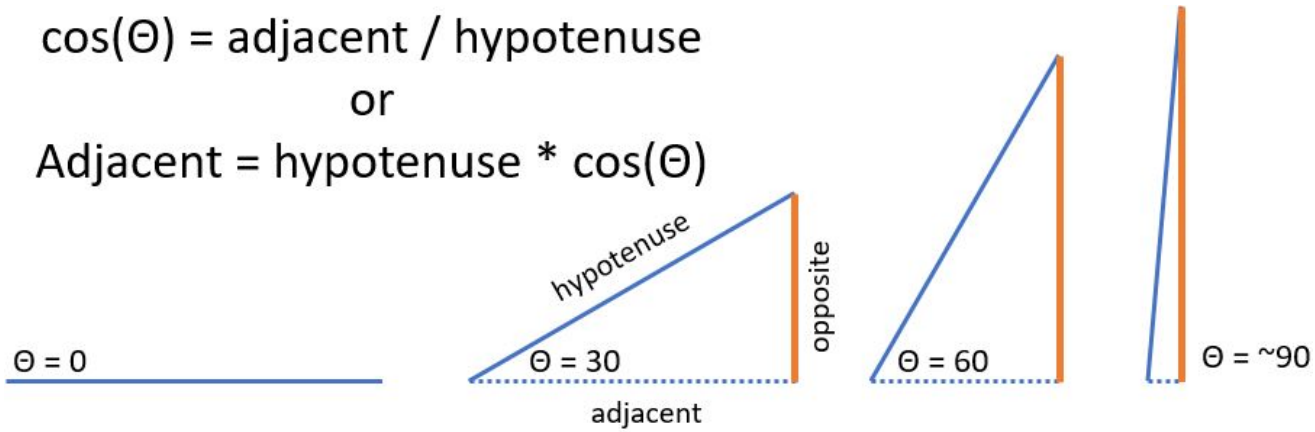
# SOH CAH TOA

- sin = opposite over hypotenuse
- cos = adjacent over hypotenuse
- tan = opposite over adjacent

cos(Θ) = adjacent / hypotenuse

or

Adjacent = hypotenuse * cos(Θ)



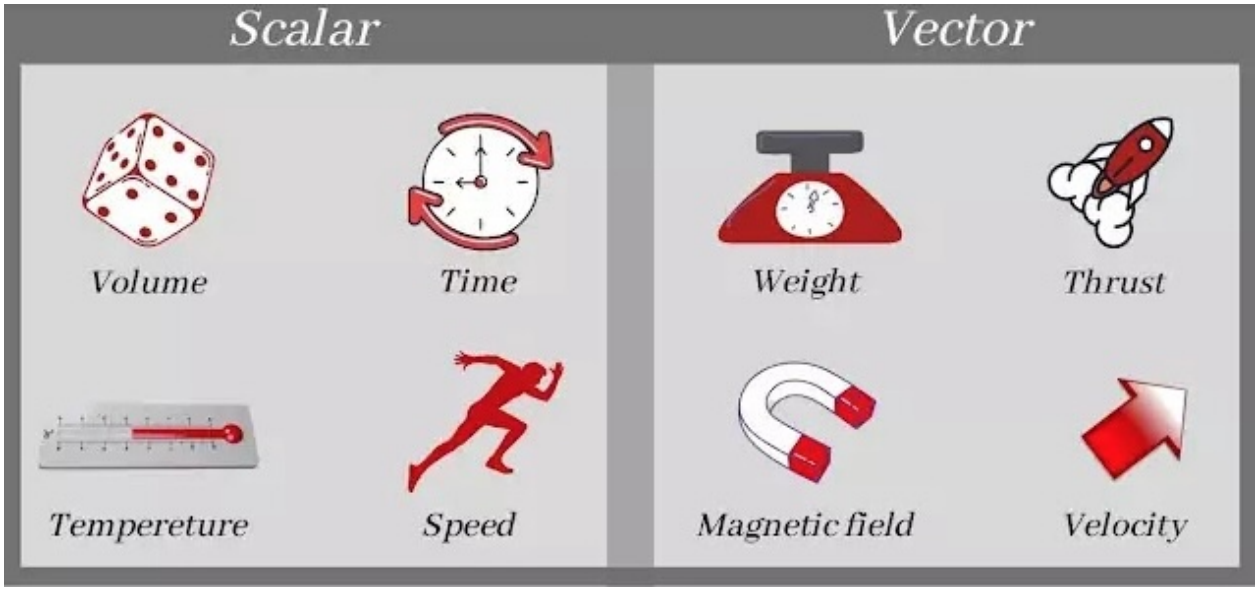

sin(Θ) = opposite / hypotenuse

or

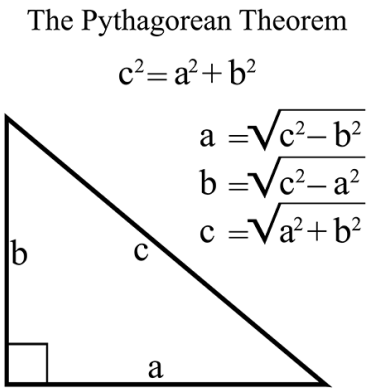
opposite = hypotenuse * sin(Θ)

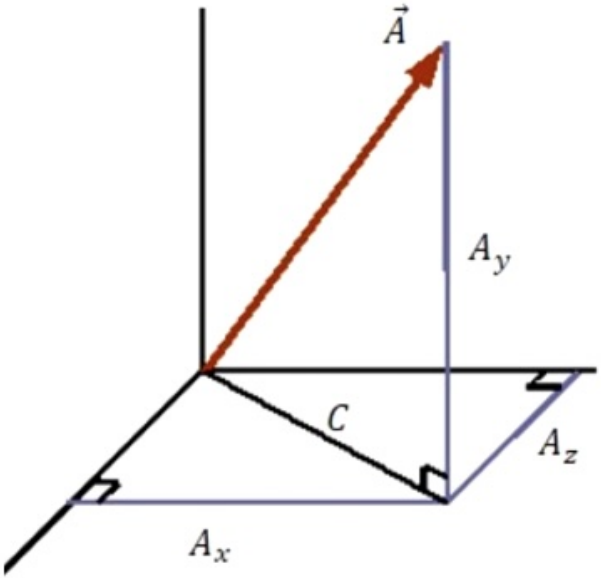
# Scalars and Vectors

Scalars are quantities that are fully described by a magnitude (or numerical value) alone.
Vectors are quantities that are fully described by both a magnitude and a direction.



# Pythagorean Theorem in 3 Dimensions

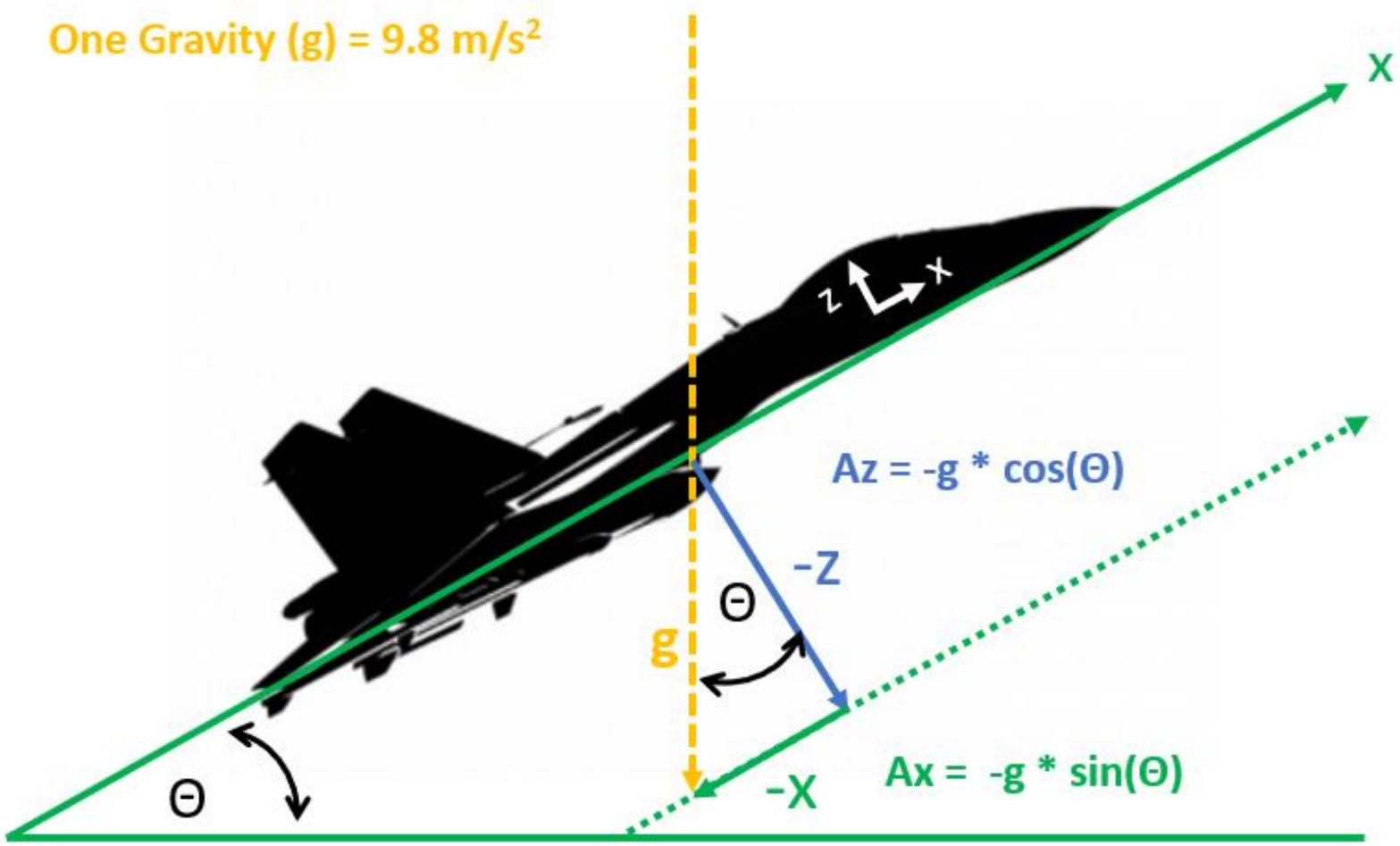


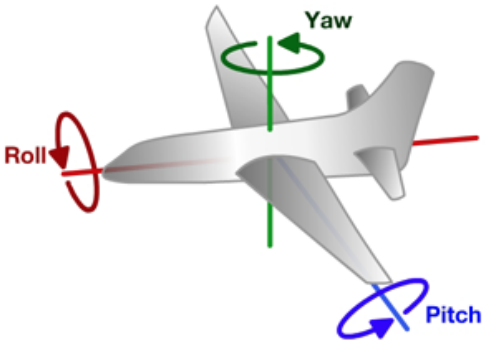

To add orthogonal (at right angles to each other) vectors in 3 Dimensions:

- $C = \sqrt{A_x^2 + A_y^2}$
- $A_{total} = \sqrt{C^2 + A_z^2}$
- $A_{total} = \sqrt{A_x^2 + A_y^2 + A_z^2}$

# Gravity and Orientation



# Assignment: L13_Motion

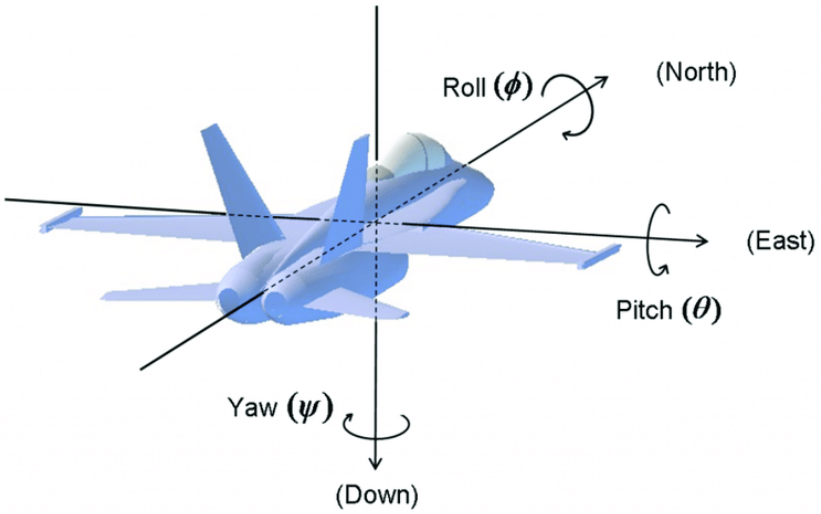



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

For the below, recall that trigonometric functions return radians which needs to be converted to degrees. See the Unit Circle slide in L02_HelloLED.

1. L13_03_Airplane
   - Calculate pitch $\theta = -asin(a_x)$.
   - Calculate roll $\phi = atan2(a_y, a_z)$.
2. L13_04_Shock
   - Store $a_{tot}$ in an array every 10ms for 5s.
   - Find and print the max value from the array.
   - Repeat.
   - Note: When at rest, the $a_{tot} \approx 1$.

# Pitch and Roll Improved

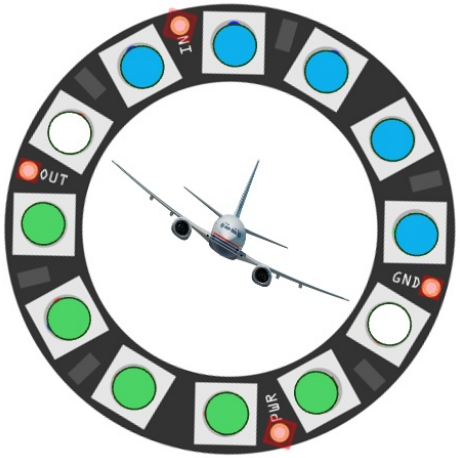



$$\text{Pitch}(\Theta) = \arctan\left(\frac{a_x}{\sqrt{{a_y}^2+{a_z}^2}}\right)$$

$$\text{Roll}\ (\Phi) = \arctan\left(\frac{a_y}{\sqrt{{a_x}^2+{a_z}^2}}\right)$$

$$\text{Yaw}\ (\Psi) = \arctan\left(\frac{\sqrt{{a_x}^2+{a_y}^2}}{{a_z}^2}\right)$$

# Assignment: L13_Motion (Extra Credit)

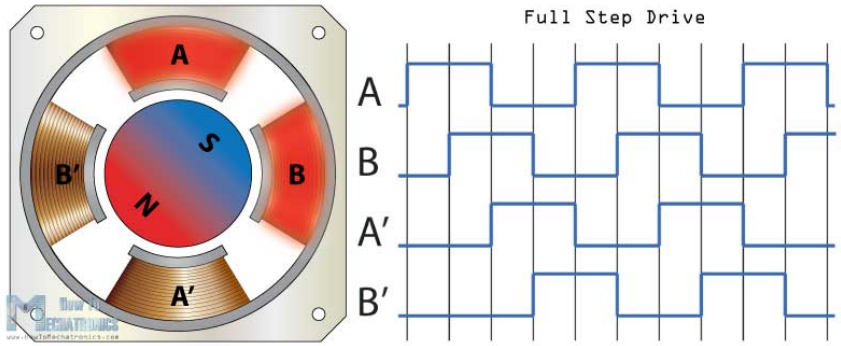




1. Modify L13_03_Airplane
   - Improve L13_03_Airplane with the more detailed trigonometric equations.
   - Use the NeoPixel ring to visualize roll.
   - When holding the accelerometer a fixed distance ($\approx 6in$) from the NeoPixel tower, change pitch and light up the pixel it is pointing at.
   - Note: for the $3^{rd}$ bullet use TOA from SOH-CAH-TOA

# Stepper Motors

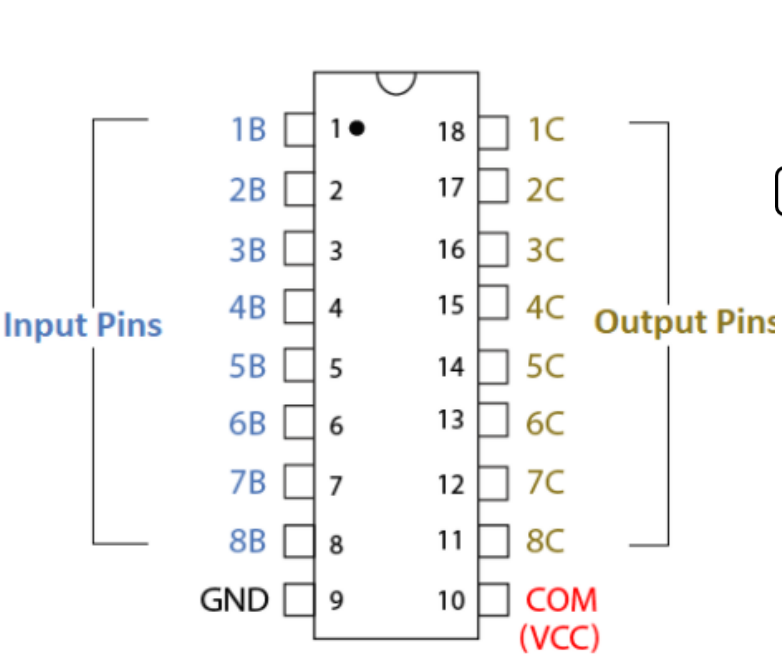
## 28BYJ Stepper Motor

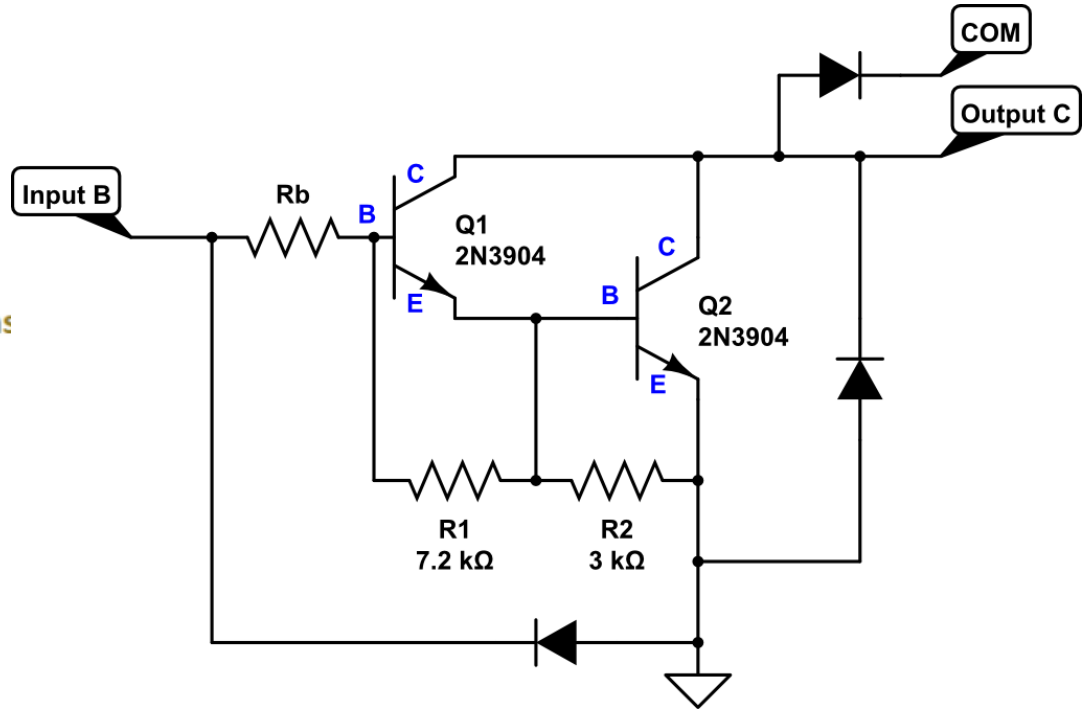
- 2048 steps per revolution
  - 32 steps per rotor revolution
  - Gear ratio 1:64
- Capable of 10-15 RPM (at 5V)
- ULN2003 Darlington Array




```
#include "Stepper.h" // Use the command palette to install the Stepper library

/* Stepper Object
  SPR is the motors steps per rotation
  IN1,IN2,IN3,IN4 are the pins connected to the motor control board's IN pins
*/
Stepper myStepper(SPR,IN1,IN3,IN2,IN4);

void setup() {
  myStepper.setSpeed(speed); // speed is an integer specifying RPM
}

void loop() {
  myStepper.step(steps); // steps is a signed integer indicating how many steps to move
}
```

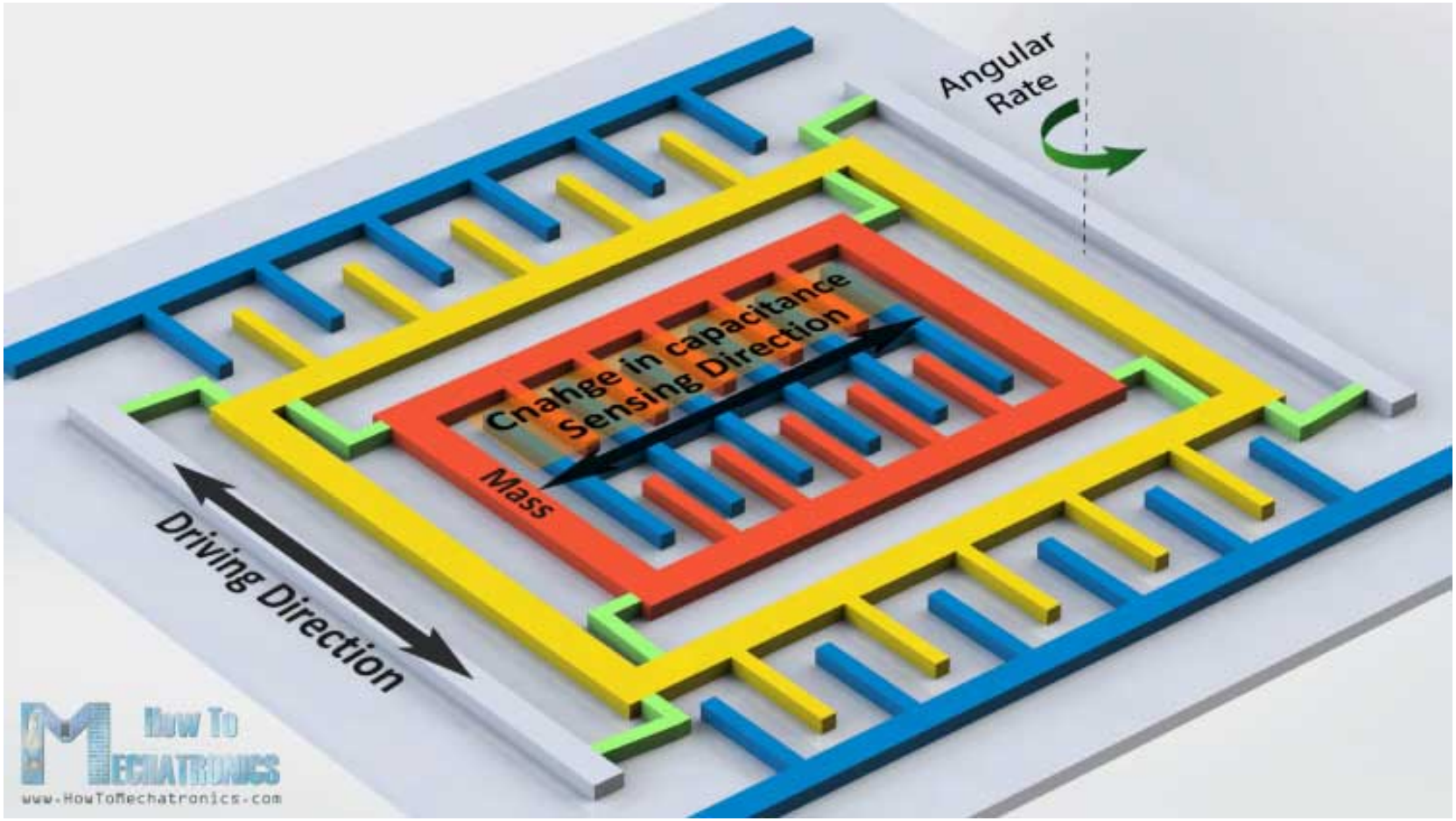
# ULN2003 Darlington Array



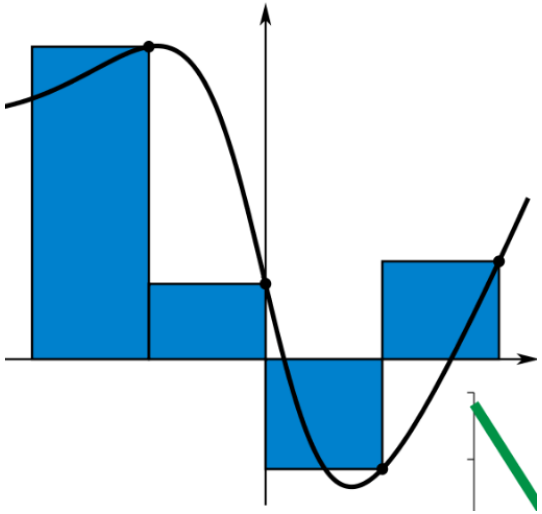

A Darlington Array is a set of current amplifying circuits that take outputs from the microcontroller and boost the current used to drive the motor.

# Gyroscopes




The gyroscope measures angular velocity (degrees per second).

# Integration via Reimann Sum



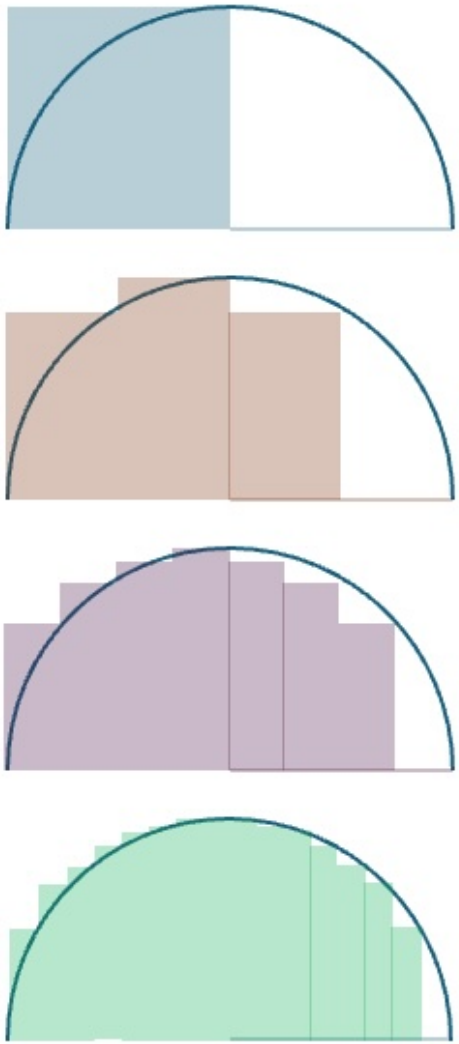
The Reimann[a] Sum can be used to find the "area under a curve":

- Divide the curve into segments vertically
- Find the point where the curve meets the right side of the first segment
- Draw a horizontal line from that point to the left side of the segment
- Shade in the resulting rectangle
- Repeat for all segments
- Sum up the area of all the rectangles[b]

[a] Georg Friedrich Bernhard Riemann was a German mathematician from the mid-1800's

[b] This is one of many ways to create the shaded area

# Reimann Sum Example

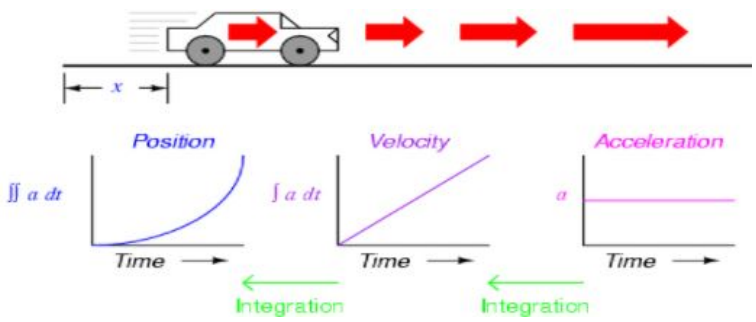


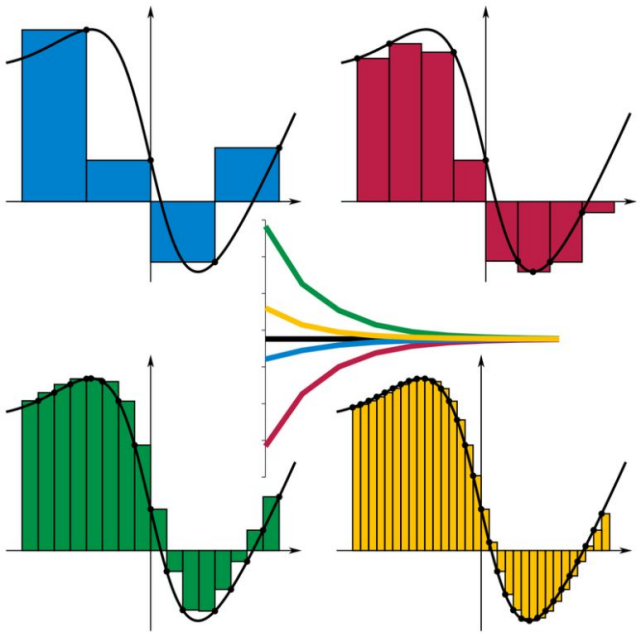
Take as an example, a half circle with a radius of 2 (area $= 2\pi = 6.28$). It can be divided into rectangular segments. Adding the areas approximates the area of the half circle.

The more segments, the more accurate:

- 2-segments: 4
- 4-segments: 5.46
- 8 segments: 5.99
- 16-segments: 6.18

# Acceleration, Velocity, and Position



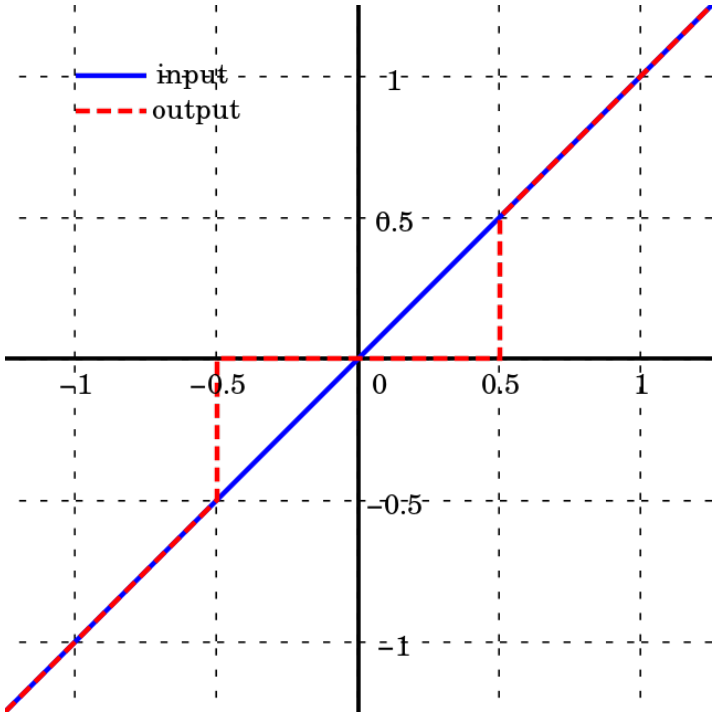



The Reimann Sum method can be used to "integrate" acceleration to velocity and velocity to position.

- Gyroscope output is angular velocity: $\omega$ ($\frac{degrees}{second}$)
- To get change in angular position ($\Delta\theta$), multiple each angular velocity by the time step: $\Delta\theta = \omega * \Delta t$
- Use the Reimann Sum to get the resulting angular position

$$\theta = \sum_{t=0}^{t=T} (\omega * \Delta t)$$

# Deadband



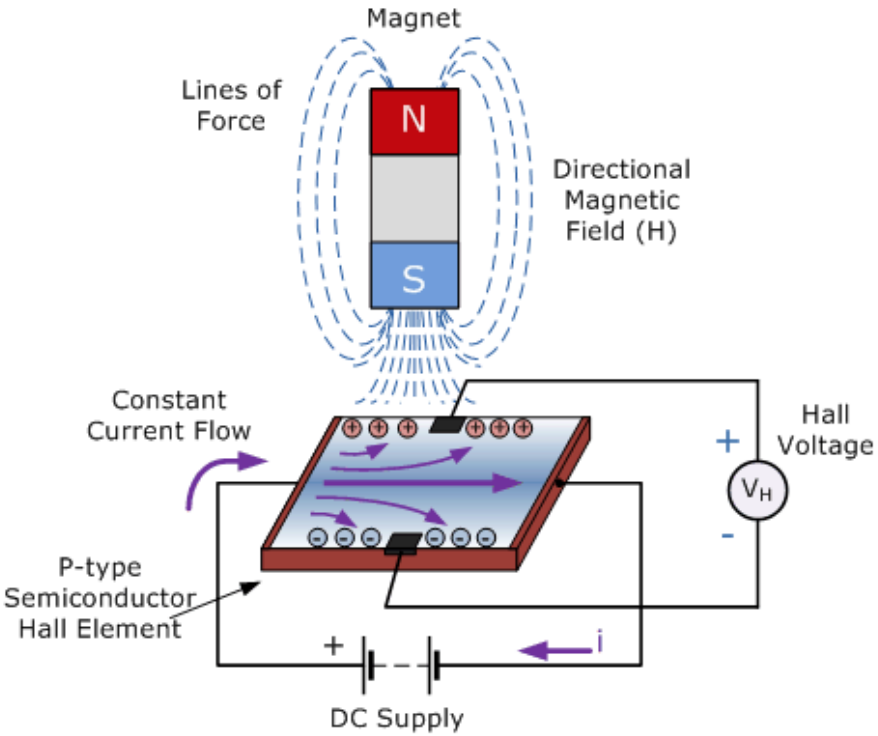

A deadband is a band of input values that in a control system create an output that is zero.

# Assignment: L13_Motion

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L13_05_DuckGoRound
   - Place your duck onto the stepper motor.
   - Wire the stepper motor noting the order: IN1, IN3, IN2, IN4.
   - In void loop(), move the motor 2 rotations clockwise, pause, 1 rotation counter-clockwise, repeat.
2. L13_06_DriveByWire
   - Connect the MPU-6050 to your system.
   - Obtain the z-axis rotation from the appropriate register on the MPU-6050. Convert to angular rotation (°per sec).
   - Calculate the angular position (°) of the gyroscope.
   - Have the stepper motor track movement in the gyroscope.

# Hall Effect Sensor




Discovered by Edwin Hall in 1879, the Hall Effect is the production of a voltage difference (the Hall voltage) across an electrical conductor, transverse to an electric current in the conductor and to an applied magnetic field perpendicular to the current.

# Interrupts

Interrupts are a way to write code that is run when an external event occurs. As a general rule, interrupt code should be very fast, and non-blocking. This means performing transfers, such as I2C, Serial, TCP should not be done as part of the interrupt handler. Rather, the interrupt handler can set a variable which instructs the main loop that the event has occurred.

```
1 pinMode(pin, INPUT); \\ can also use INPUT_PULLUP or INPUT_PULLDOWN
2 attachInterrupt(pin, function, mode);
```

Mode: defines when the interrupt should be triggered. Three constants are predefined as valid values:

- CHANGE to trigger the interrupt whenever the pin changes value,
- RISING to trigger when the pin value goes from low to high,
- FALLING for when the pin value goes from high to low.

# Software Timers

There are also software timer interrupts. The Photon2 can manage up to 10 timers simultaneously.

```
1  Timer timer(1000,printEverySecond);
2
3  void setup() {
4    Serial.begin(9600);
5    timer.start();
6  }
7
8  void printEverySecond() {
9      static int count = 0;
10     Serial.printf("count=%i, time = %u ms \n", count,millis());
11     count++;
12 }
```

Note:

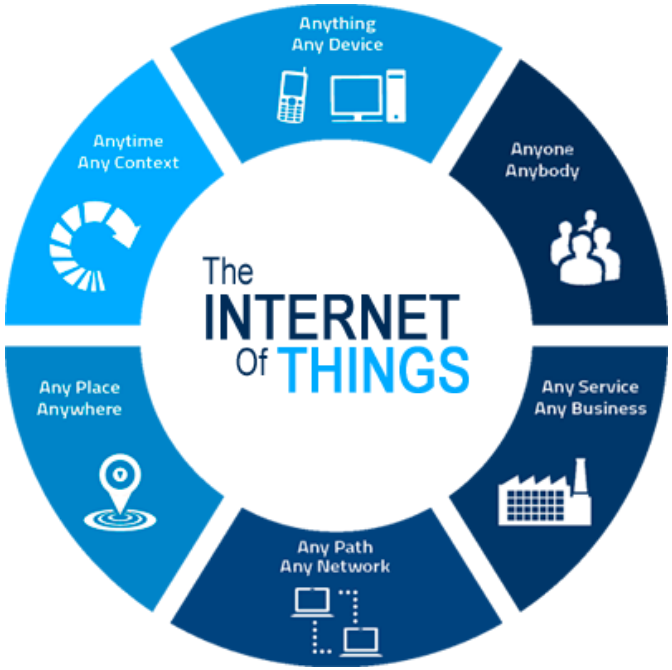
- The timer callback is similar to an interrupt - it shouldn't block.
- Multiple timers are serviced sequentially when several timers trigger simultaneously, thus requiring special consideration when writing callback functions.

# Assignment: L13_Motion

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L13_07_Alarm
   - Connect Hall Effect Sensor, button, and Neopixel to simulate an alarm system.
   - Use the button to enable / disable alarm.
   - Light up NeoPixels:
     - BLUE when alarm disarmed
     - GREEN when armed and magnet detected.
     - Blinking RED when armed and magnet is not detected.
2. L13_08_RPM
   - Place magnet on shaft of a drill
   - Create an interrupt function that returns the time per rotation using the Hall Effect Sensor.
   - Convert this time to rotations per minute.
   - Display on Adafruit.io databoard.
   - EXTRA:
     - Create a speedometer using a servo motor.

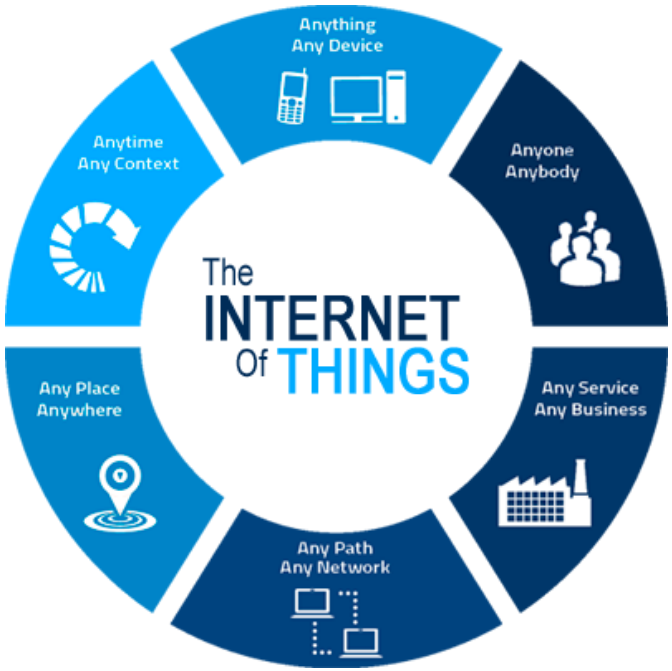
# Module 13 Review




- Learning Objectives
  1. Utilized a Data Sheet to learn to access a component
  2. Accelerometers and Gyroscopes
  3. A little big of trig and calculus
  4. Stepper motors
  5. Hall effect sensors
  6. Interrupts
- Additional Items
  1. Quiz 9

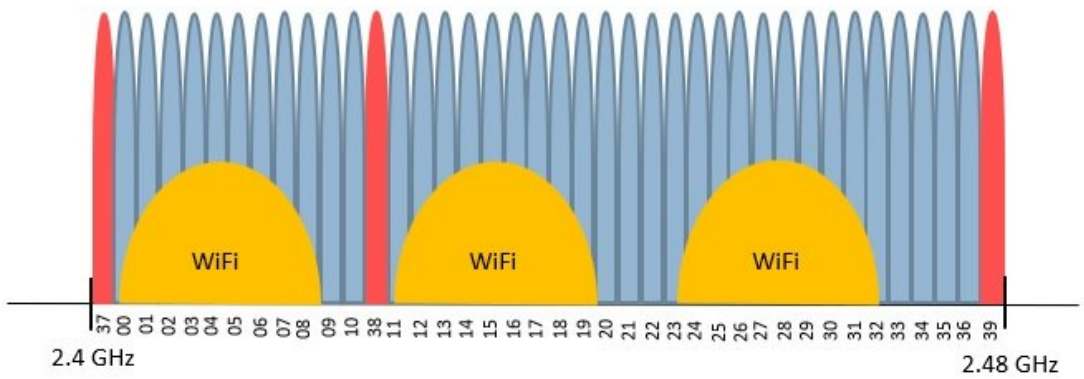
# Module 14 - Radio

# Module 14 Objectives



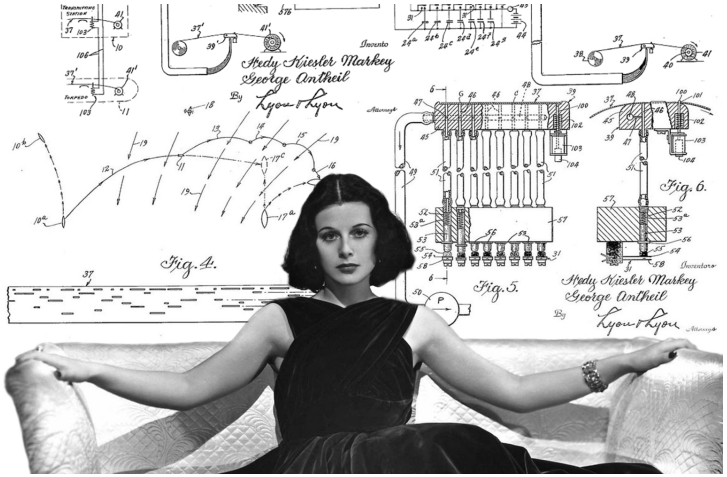

- Learning Objectives
  1. Bluetooth
  2. GPS
  3. UART / SPI
  4. LoRa
- Additional Items
  1. 3D Modeling Lesson 6 - ESP32-CAM
  2. Quiz 10

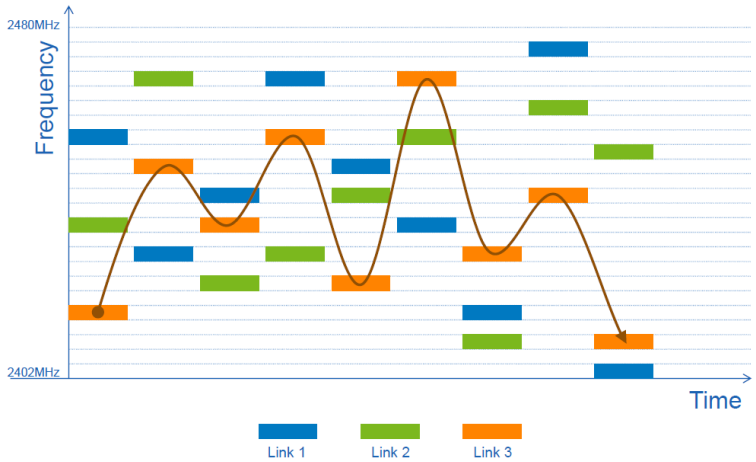
# Bluetooth




- The Bluetooth protocol operates at 2.4GHz in the same unlicensed ISM frequency band where RF protocols like ZigBee and WiFi also exist.
- Bluetooth networks (commonly referred to as piconets) use a master/slave model to control when and where devices can send data. In this model, a single master device can be connected to up to seven different slave devices. Any slave device in the piconet can only be connected to a single master.

# Frequency Hopping




Hedy Lamarr (actress/inventor)

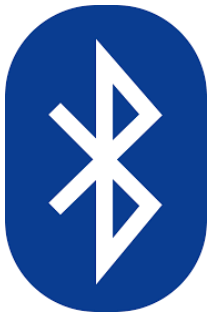
- Starred in over 35 movies
- Invented Frequency Hopping[a]
- "Mother of WiFi"

[a] with composer George Antheil

Frequency Hopping

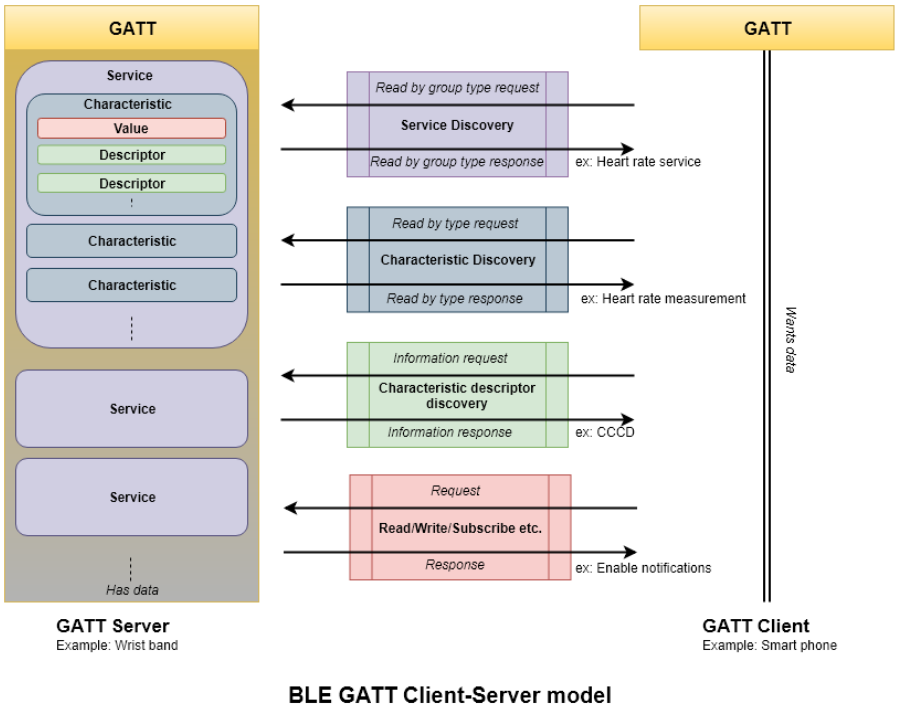
- Rapidly change frequencies
- Uses pseudo-random sequence
- Avoids interference

# Bluetooth - Generic Access Profile



- The Generic Access Profile (GAP) controls connections and advertising in Bluetooth. GAP is what makes your device visible to the outside world, and determines how two devices can (or can't) interact with each other.
- GAP defines various roles for devices, but the two key concepts to keep in mind are Central devices and Peripheral devices.
  - Peripheral devices are small, low power, resource constrained devices that can connect to a much more powerful central device. Peripheral devices are things like a heart rate monitor, a BLE enabled proximity tag, etc.
  - Central devices are usually the mobile phone or tablet that you connect to with far more processing power and memory.

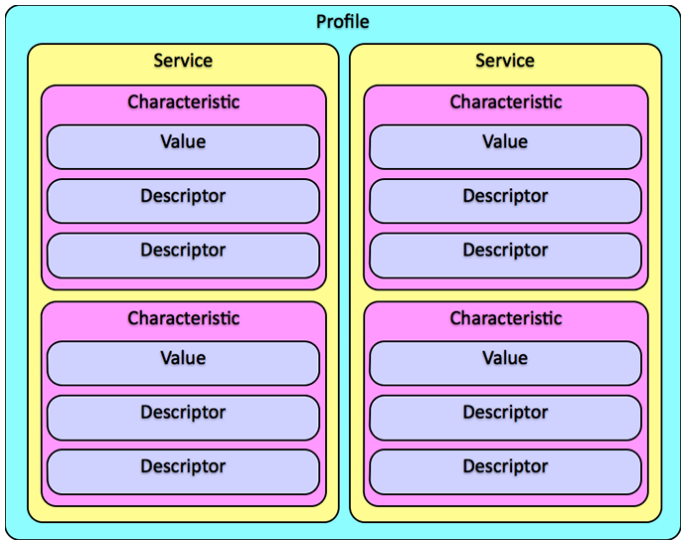
# Bluetooth - Generic Attribute Profile (GATT)




BLE GATT Client-Server model

- Generic Attribute Profile defines the way that two BLE devices transfer data back and forth using concepts called Services and Characteristics. It makes use of a generic data protocol called the Attribute Protocol (ATT), which is used to store Services, Characteristics and related data in a simple lookup table using 16-bit IDs for each entry in the table.
- GATT comes into play once a dedicated connection is established between two devices, meaning that you have already gone through the advertising process.

# Bluetooth - Services and Profiles




- A Profile is a pre-defined collection of Services. The Heart Rate Profile, for example, combines the Heart Rate Service and the Device Information Service.
- Services break data up into logic entities, and contain specific chunks of data called characteristics. A service can have one or more characteristics, and each service distinguishes itself from other services with a unique numeric ID called a UUID, which can be either 16-bit (official BLE Services) or 128-bit (custom services).
- A Characteristic contains a single data point or an array of related data. For example: X/Y/Z values of an accelerometer.

# ASCII Reminder

- ASCII characters (the symbols that we are use to reading) can be represented by a single byte (uint8_t).

## ASCII TABLE

| Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | [NULL] | 32 | 20 | [SPACE] | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 1 | [START OF HEADING] | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 2 | [START OF TEXT] | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 3 | [END OF TEXT] | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 4 | [END OF TRANSMISSION] | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 5 | [ENQUIRY] | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 6 | [ACKNOWLEDGE] | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 7 | [BELL] | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 8 | [BACKSPACE] | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 9 | [HORIZONTAL TAB] | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | A | [LINE FEED] | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | B | [VERTICAL TAB] | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | C | [FORM FEED] | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | D | [CARRIAGE RETURN] | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | E | [SHIFT OUT] | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | F | [SHIFT IN] | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | [DATA LINK ESCAPE] | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | [DEVICE CONTROL 1] | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | [DEVICE CONTROL 2] | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | [DEVICE CONTROL 3] | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | [DEVICE CONTROL 4] | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | [NEGATIVE ACKNOWLEDGE] | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | [SYNCHRONOUS IDLE] | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | [ENG OF TRANS. BLOCK] | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | [CANCEL] | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | [END OF MEDIUM] | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | [SUBSTITUTE] | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | [ESCAPE] | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | [FILE SEPARATOR] | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | [GROUP SEPARATOR] | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | [RECORD SEPARATOR] | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | [UNIT SEPARATOR] | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | [DEL] |

ASCII: American Standard Code For Information Interchange

# Photon2 BLE - UART Service

```
1  // These UUIDs were defined by Nordic Semiconductor and are now the defacto standard for
2  // UART-like services over BLE. Many apps support the UUIDs now, like the Adafruit
       Bluefruit app.
3  const BleUuid serviceUuid("6E400001-B5A3-F393-E0A9-E50E24DCCA9E");
4  const BleUuid rxUuid("6E400002-B5A3-F393-E0A9-E50E24DCCA9E");
5  const BleUuid txUuid("6E400003-B5A3-F393-E0A9-E50E24DCCA9E");
6
7  BleCharacteristic txCharacteristic("tx", BleCharacteristicProperty::NOTIFY, txUuid,
       serviceUuid);
8  BleCharacteristic rxCharacteristic("rx", BleCharacteristicProperty::WRITE_WO_RSP, rxUuid,
        serviceUuid, onDataReceived, NULL);
9  BleAdvertisingData data;
10
11 //onDataReceived is used to receive data from Bluefruit Connect App
12 void onDataReceived(const uint8_t* data, size_t len, const BlePeerDevice& peer, void*
       context) {
13   uint8_t i;
14
15   Serial.printf("Received data from: %02X:%02X:%02X:%02X:%02X:%02X \n", peer.address()
       [0], peer.address()[1],peer.address()[2], peer.address()[3], peer.address()[4], peer
       .address()[5]);
16   Serial.printf("Bytes: ");
17   for (i = 0; i < len; i++) {
18     Serial.printf("%02X ",data[i]);
19   }
20   Serial.printf("\n");
21   Serial.printf("Message: %s\n",(char *)data);
22 }
```

# Photon2 BLE - UART Transmit Example

```
1  const int UART_TX_BUF_SIZE = 20;
2  uint8_t txBuf[UART_TX_BUF_SIZE];
3  uint8_t i;
4
5  SYSTEM_MODE(SEMI_AUTOMATIC); //Using BLE and not Wifi
6
7  void setup() {
8      Serial.begin();
9      waitFor(Serial.isConnected, 15000);
10
11     BLE.on();
12     BLE.addCharacteristic(txCharacteristic);
13     BLE.addCharacteristic(rxCharacteristic);
14     data.appendServiceUUID(serviceUuid);
15     BLE.advertise(&data);
16
17     Serial.printf("Photon2 BLE Address: %s\n", BLE.address().toString().c_str());
18 }
19
20 void loop() {
21     for(i=0;i<UART_TX_BUF_SIZE-1;i++) {
22       txBuf[i] = random(0x40,0x5B); //Captial ASCII characters plus @
23     }
24     txBuf[UART_TX_BUF_SIZE-1] = 0x0A;
25     txCharacteristic.setValue(txBuf, UART_TX_BUF_SIZE);
26     for(i=0;i<UART_TX_BUF_SIZE;i++) {
27         Serial.printf("%c",txBuf[i]);
28     }
29     delay(5000);
30 }
```

# sprintf(): Formatted Print to a Buffer

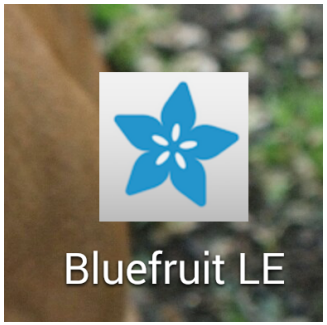
sprintf() can be used to prepare a payload of data (buf) to be transferred between devices using BLE.

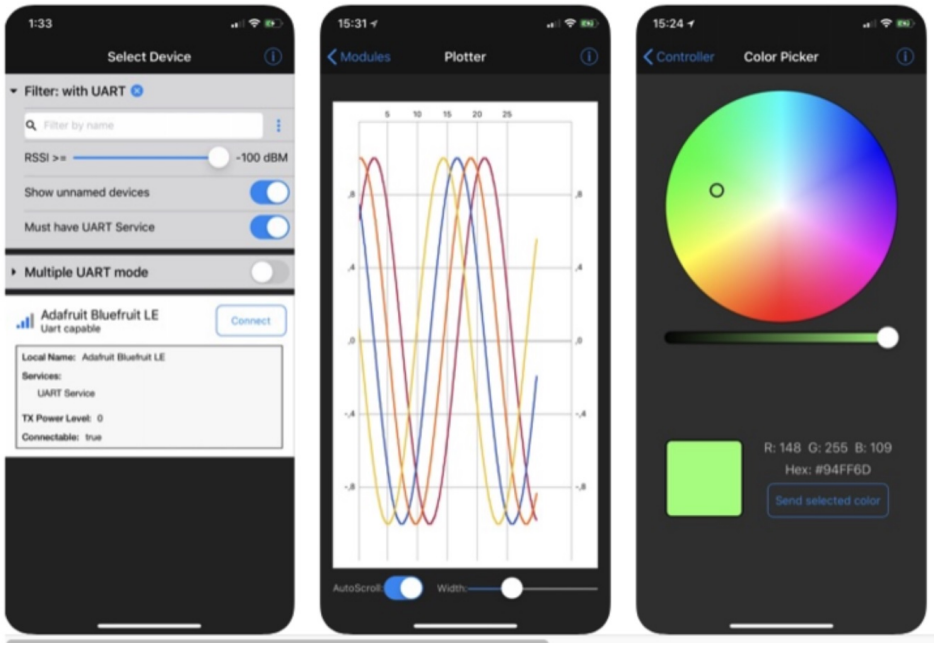
```
// sprintf does a formatted print to a buffer of type char[]

const int BUFSIZE = 50;
byte buf[BUFSIZE];
int people;
float dogs,avg;

void setup() {
  Serial.begin(9600);
  people = 5;
  dogs = 13;
}

void loop() {
  avg = dogs / people;
  sprintf((char *)buf,"The %i people have on average %0.2f dogs \n",people, avg);
  buf[BUFSIZE-1] = 0x0A; //ensure last character is a line feed (LF)
  Serial.printf("Sending 'buf' contains the string: %s",(char *)buf);
  txCharacteristic.setValue(buf, BUFSIZE);
}
```

Note: Windows treats a $\backslash n$ as a LF-CR (0x0D0A), if a LF is needed (e.g., for BLE) then it needs to be inserted manually:

```
buf[BUFSIZE-1] = 0x0A;
```

# Install Bluefruit Connect App



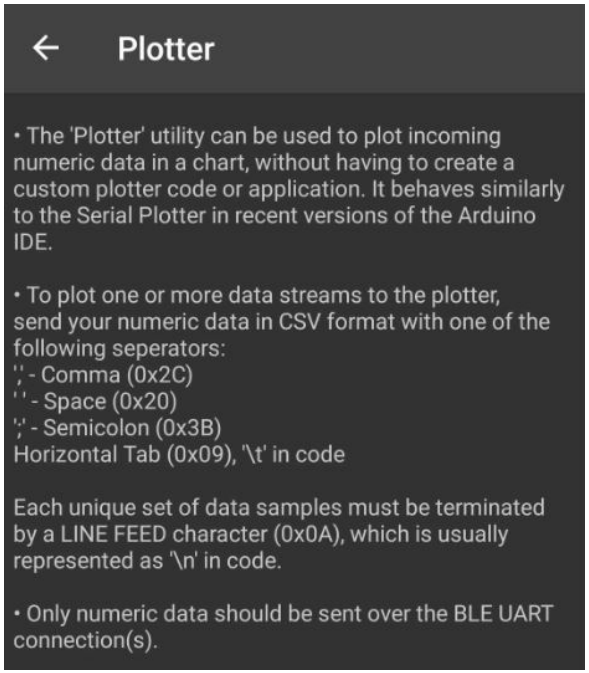




Install the Bluefruit Connect app from the App Store and Google Play

# Assignment: L14_01_BlueTooth


- Load Bluefruit Connect on your smart device. Using the code on the proceeding slides, establish and test BLE UART communications
- Attach the encoder and Neopixels to your Photon2.
- Review your NeoPixel ring/strip assignment (L05_02_NeoPixel) and copy the appropriate code to L14_01_BlueTooth
- Comment out sending a random string. Instead, when the encoder changes, send the NeoPixel position via BLE to Bluefruit Connect.
- Reset the encoder and NeoPixels to the appropriate state when a pixel number is received from Bluefruit Connect (zero to total number of pixels).
  - This can be accomplished by converting from pixel to encoder position and using myEnc.write() in onDataRecieved()

# Assignment: L14_BlueTooth - Colors



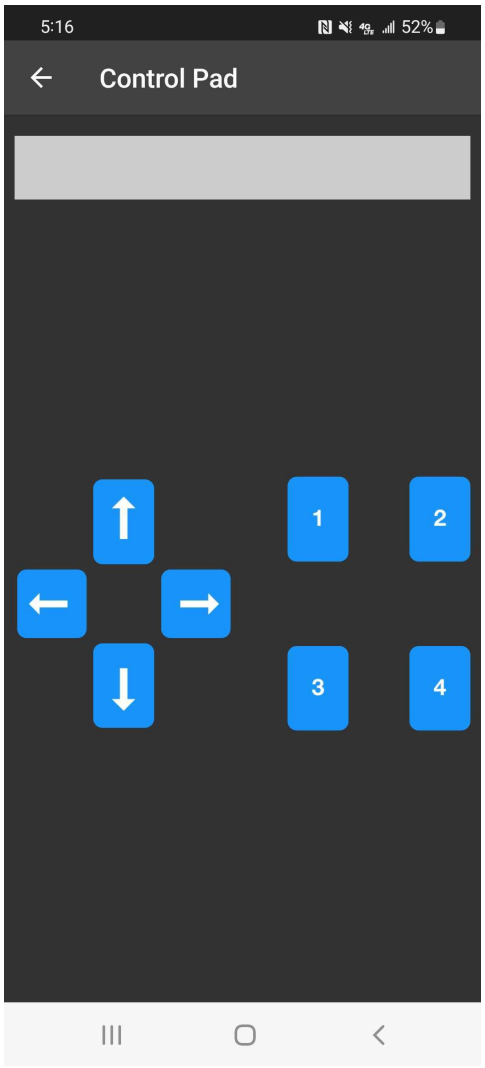

ColorPicker String
5 bytes plus CR

['!'] ['C'] [byte red] [byte green] [byte blue] [CRC]

## L14_01_BlueTooth (Continued)

- Plotter function in Bluefruit Connect:
  1. Every time the encoder moves, generate and change the pixels to a random color (R,G,B format, not Hex).
  2. Plot the pixel number and three RGB components on the Bluefruit Plotter.
- Controller -> Color Picker screen on Bluefruit Connect:
  1. Send a color to the Photon2.
  2. Identify in your code if ColorPicker string or general UART string is received.
  3. If ColorPicker, then using bitwise left shift and OR to convert string to hex color similar to L09_02_RetrieveShow
  4. Change your Neopixel color to match Color Picker

# Assignment: L14_BlueTooth - Colors (EXTRA)

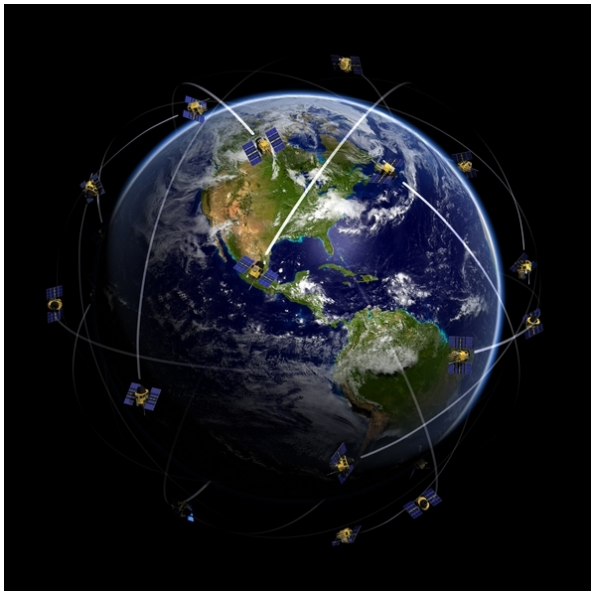


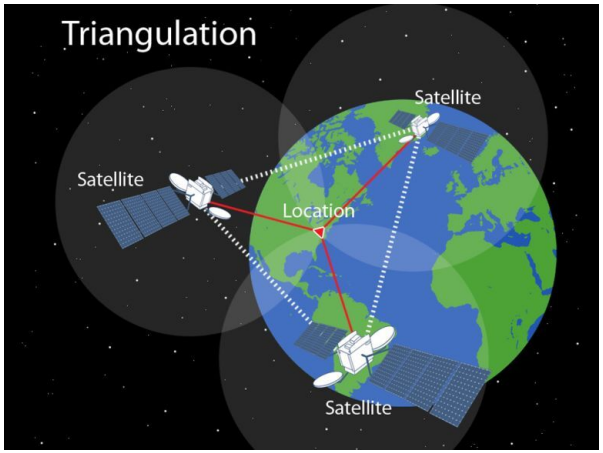

L14_01_BlueTooth (EXTRA)

- Experiment sending signals from the Bluefruit Connect Control Pad
- Use the Up/Down arrows to change the neopixel brightness
- Use the Left/Right arrows to cycle through a rainbow
- Code in a different neopixel effect for each of the number keys

# Global Positioning System

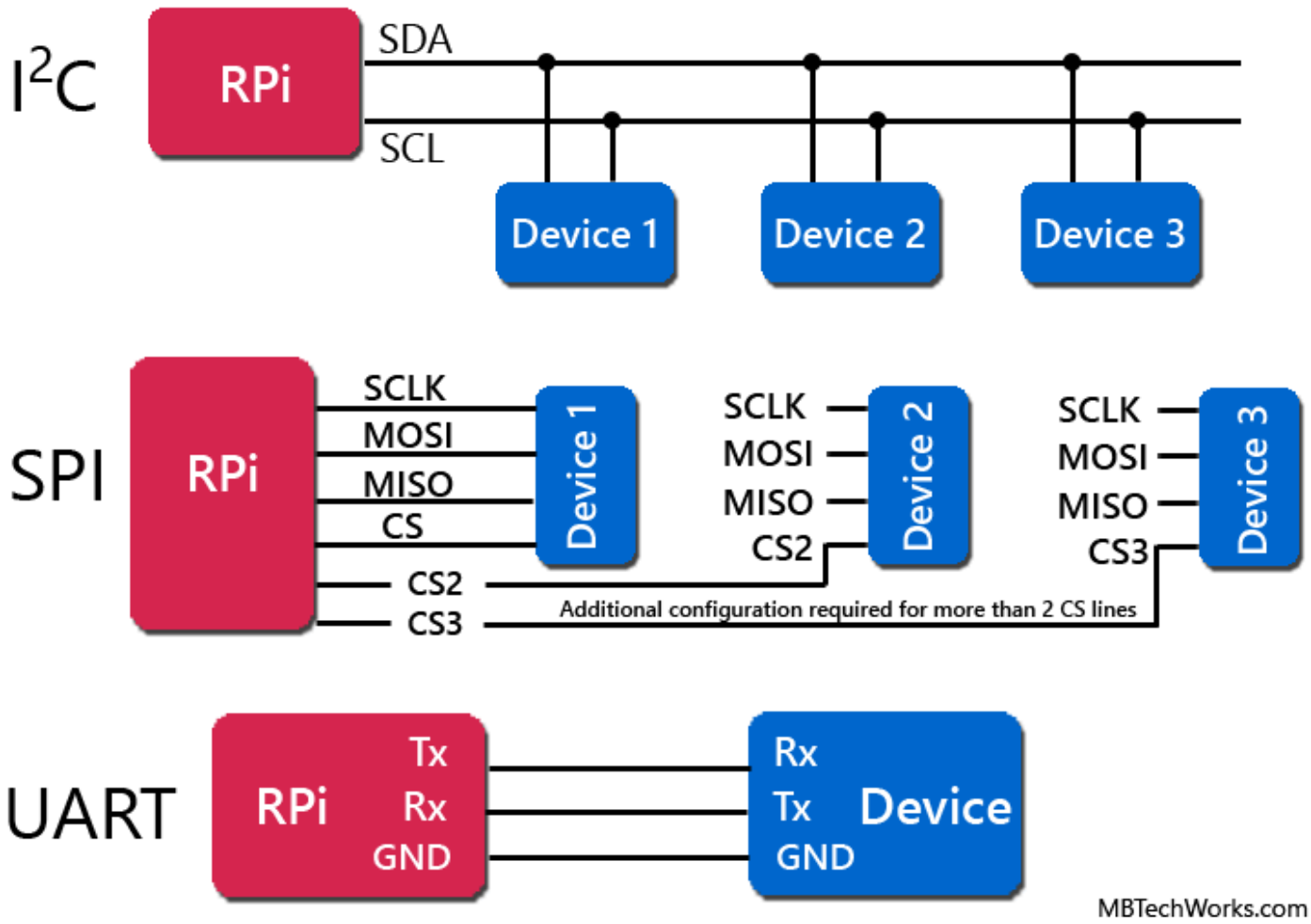
# Global Positioning System



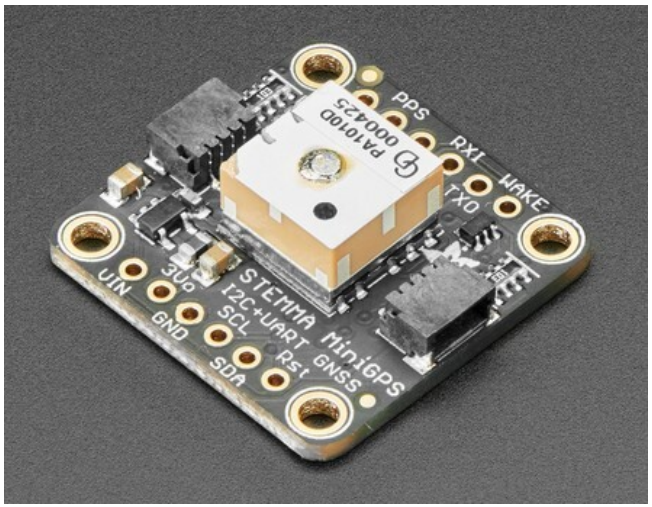



- 31 satellites at an altitude of 11000 miles, circling the earth every 718 minutes.
- Envisioned by Aerospace Corporation 1963
- First satellite 1973
- Open to commercial use 1985
- GPS receiver measures time it takes signal (at speed of light) to get from satellite to receiver.
- Uses triangulation from at least 4 satellites to obtain latitude, longitude, altitude, and time.
- GNSS is an international system of satellites that includes GPS, BDS, Galileo, GLONASS, IRNSS, QZSS and others

# UART vs I²C vs SPI



# Global Positioning System



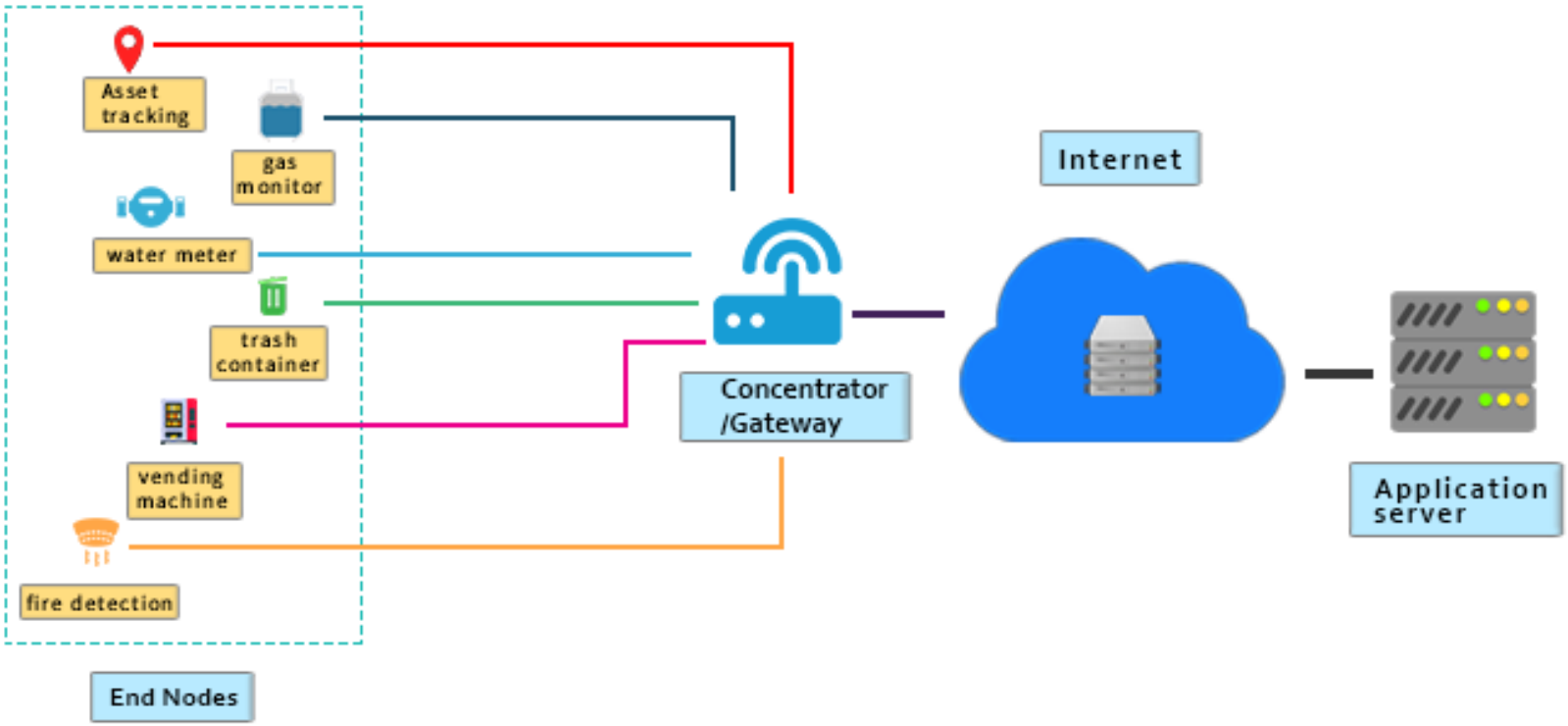

- Miniature GPS module
- Houses a complete GPS/GNSS solution
- Both I2C and UART interfaces
- STEMMA interface for easy prototyping

# Adafruit_GPS Library

```
$GPGGA,202410.000,4042.6000,N,07400.4858,W,1,4,3.14,276.7,M,-34.2,M,,*63
$GPRMC,202410.000,A,4042.6000,N,07400.4858,W,0.08,161.23,160412,,,A*70
$GPGGA,202411.000,4042.5999,N,07400.4854,W,1,3,17.31,275.8,M,-34.2,M,,*5D
$GPRMC,202411.000,A,4042.5999,N,07400.4854,W,0.14,161.23,160412,,,A*7A

Time: 20:24:11.0
Date: 16/4/2012
Fix: 1 quality: 1
Location: 4042.5998N, 7400.4853W
Speed (knots): 0.14
Angle: 161.23
Altitude: 275.80
Satellites: 3
```

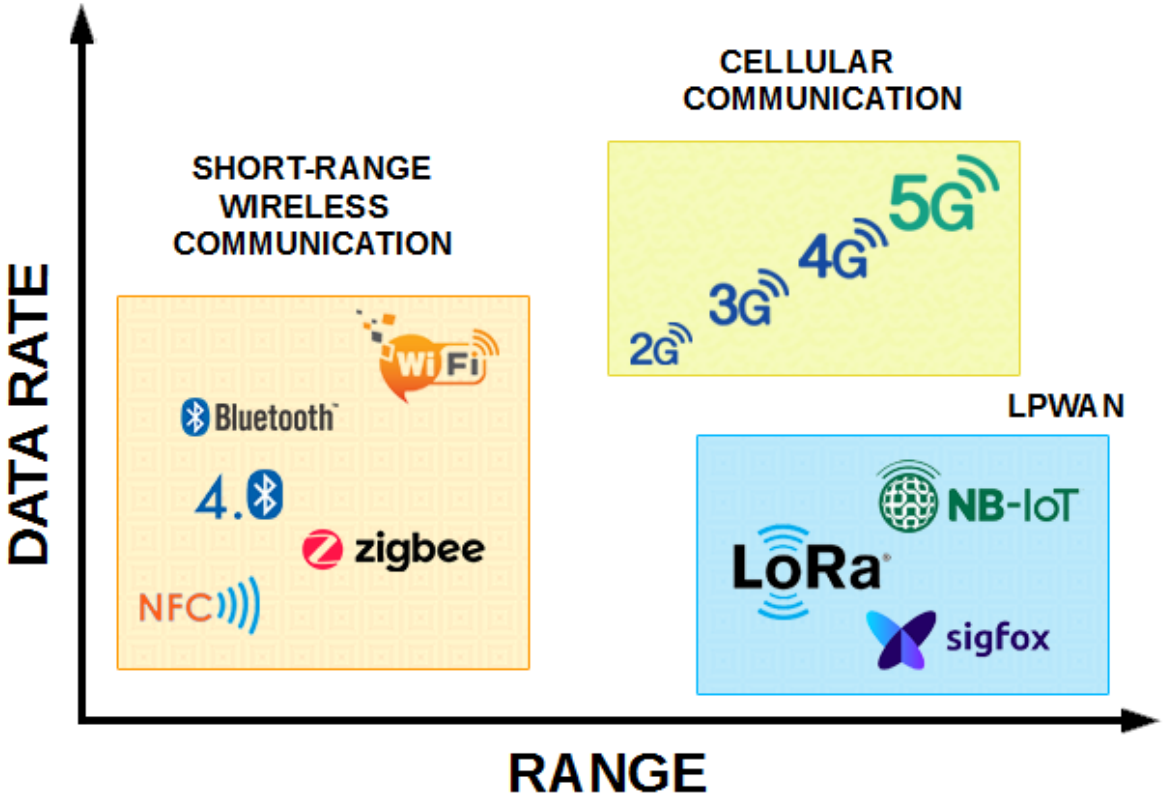
- Call gps.read() at the beginning of void loop()
- Then immediately call GPS.parse(GPS.lastNMEA()) to make the data available
- When you need it, you can than access GPS.latitude, GPS.longitude, GPS.speed, etc.

# LoRa




LoRa is a long range, low power, inexpensive technology for Internet of Things

# LoRa Range vs Data Rate



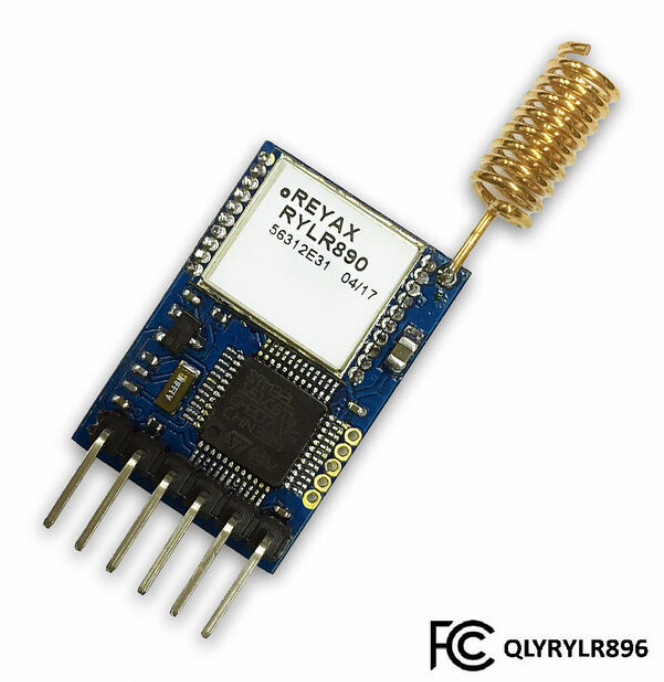

LoRa uses license-free sub-gigahertz radio frequency ISM bands in the deployed region such as 868 MHz in Europe and 915MHz in North America.

# LoRa Features

LoRa was many desirable features:

- It has very wide coverage range about 5 km in urban areas and 15 km in suburban areas
- Battery lifetime up to 15 years
- One LoRa gateway takes care of thousands of nodes.
- Easy to deploy and low cost.
- Enhanced the secure data transmission by embedded end-to-end AES128 encryption

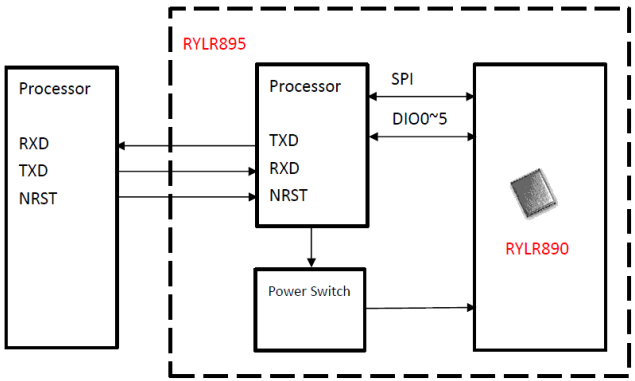
# RXLR896 LoRa Module




Features:

- Semtech SX1276 Engine
- Excellent blocking immunity
- Low receive current
- High sensitivity
- Control easily by AT commands
- 127 dB Dynamic Range RSSI
- Designed with integrated antenna
- AES128 Data encryption

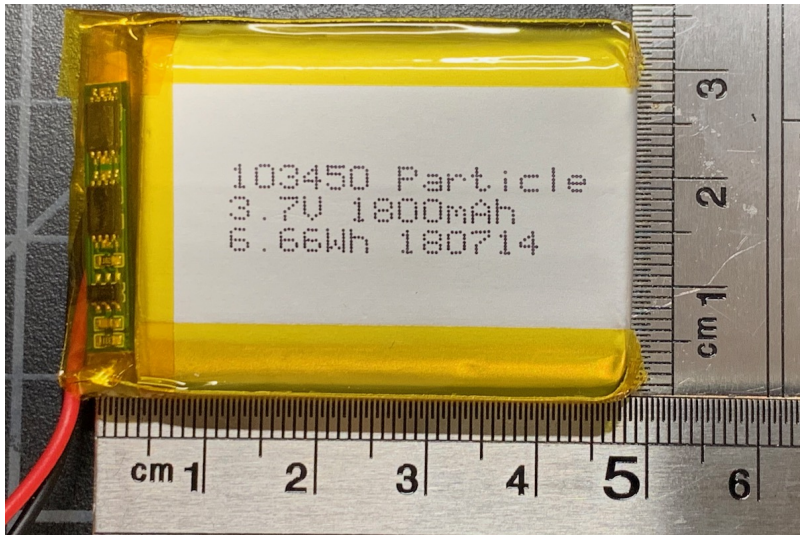
# AT Commands




AT commands are commands which are used to control the modems where AT stands for Attention. These commands were derived from Hayes commands which were used by the Hayes smart modems. Every wireless modem requires an AT command to interact with a computer machine.

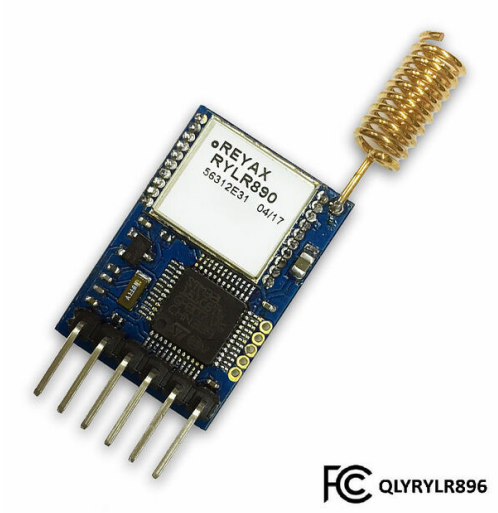
Communication between the Photon2 and RYLR896 takes place over UART (Serial1) using the same "Serial" commands that were used in Lesson 3 and Lesson 4.

# Batteries - Going Mobile




1. All Particle platforms have JST-PH pins for a Lithium Polymer (LiPo) battery
   - Always check wiring polarity
2. Battery can be charged via USB port or $V_{bus}$.
3. Battery power is available on LiPo+ or 3.3V pins

# Assignment: LoRaGPS

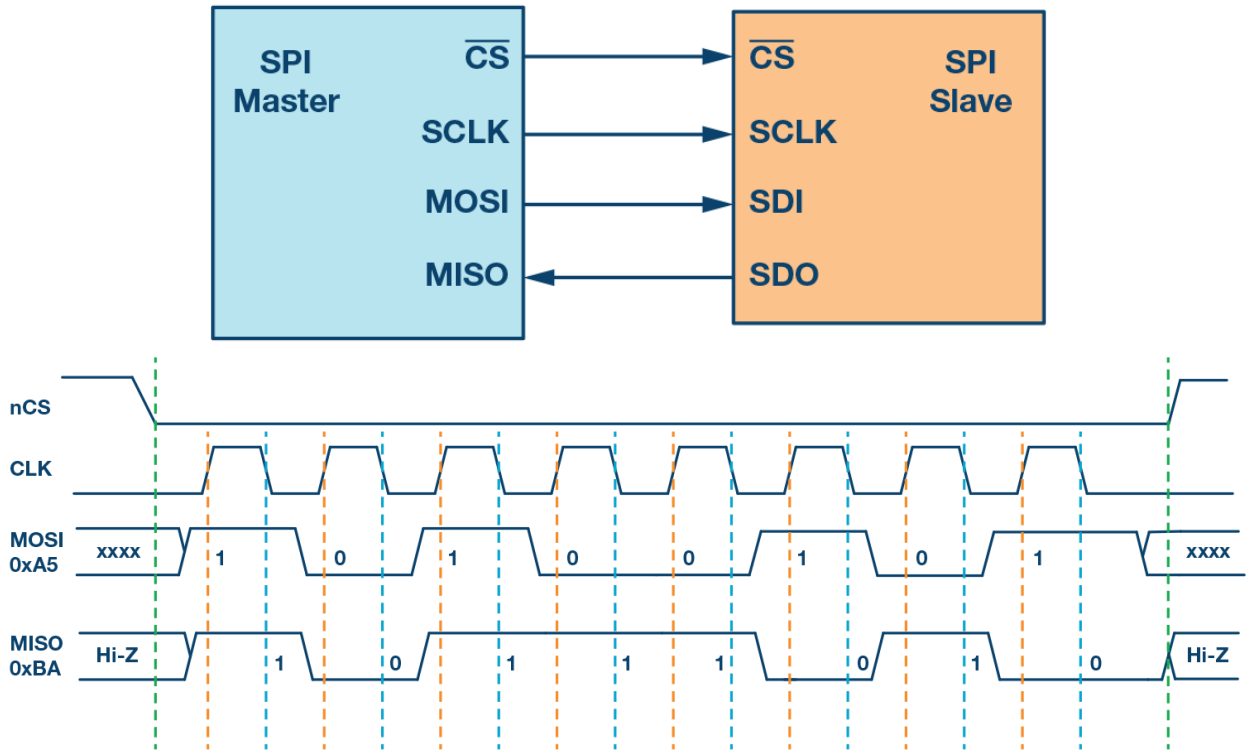



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

Features:

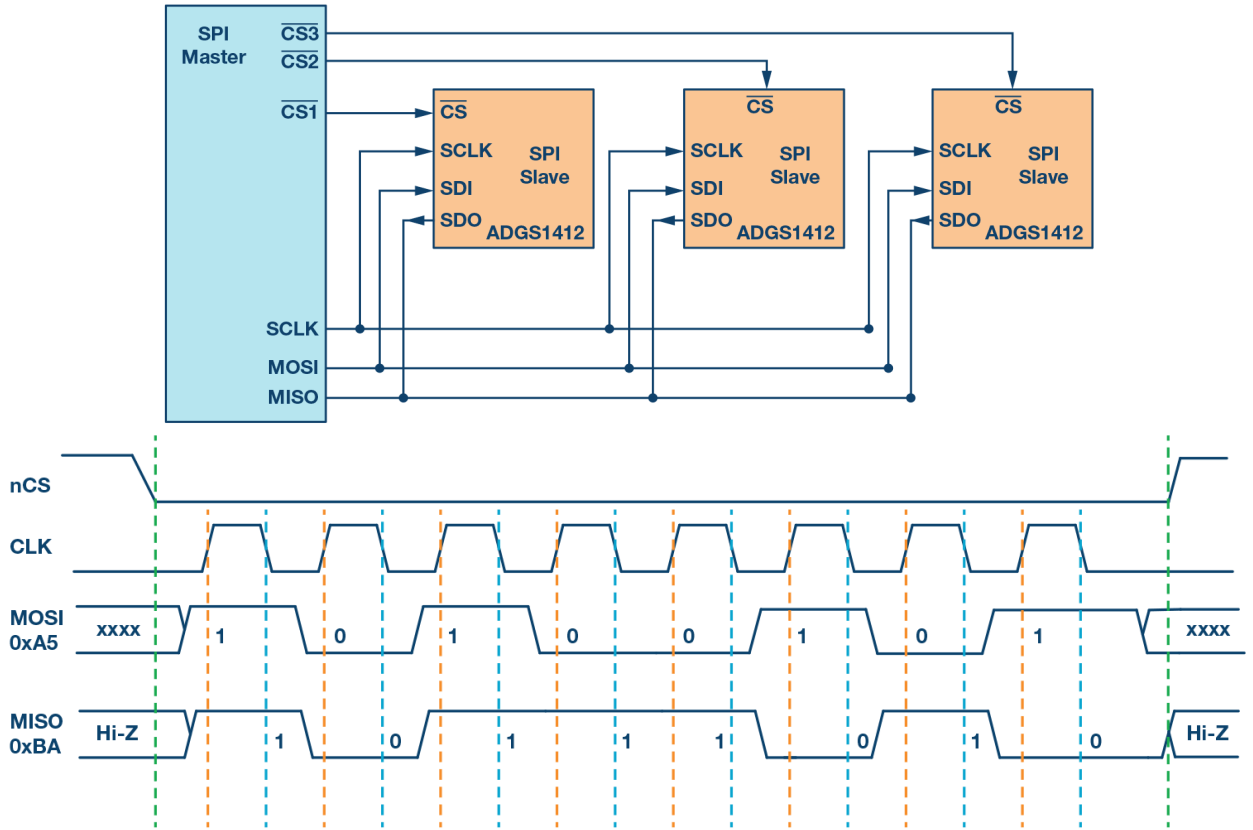
1. Borrow a battery, GPS unit, and LoRa unit from the instructor
2. Using the L14_00_GPS code, obtain GPS coordinates and display on OLED
3. Modify L14_04_LoRaGPS:
   - Integrate the LoRa, GPS, and OLED
   - Get your own RADIOADDRESS from instructors
   - Using IoT_Timer, turn on the D7 LED for 5 seconds when LoRa data received.
   - Display FUSE Sound/Particles to OLED
   - Send live GPS coordinates back to LoRa base-station
   - Find the distance you can go N/S/E/W and get a LoRa signal back to FUSE.
   - Class Trip: repeat at ABQ Biopark

# Serial Peripheral Interface



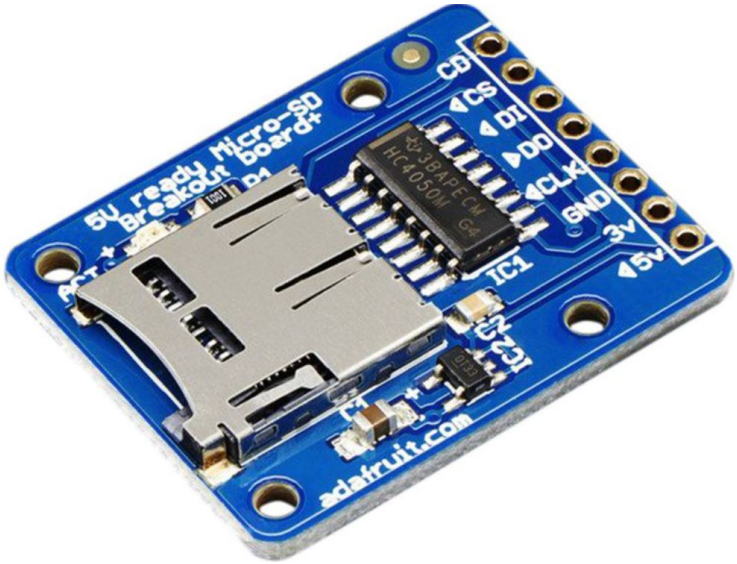

- Master Out, Slave In (MOSI) connects to Data In
- Master In, Slave Out (MISO) connects to Data Out

# Multiple SPI Modules



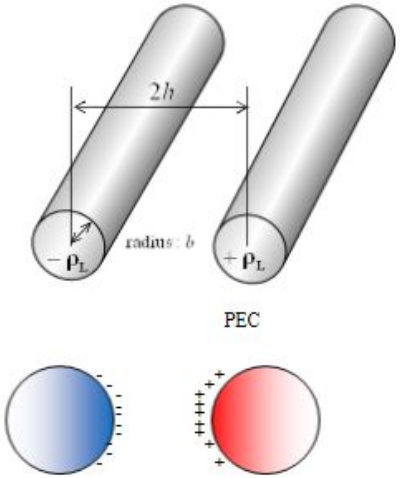

- SPI uses the $\overline{CS}$ lines to select which peripheral is active.
- Having two SPI devices selected at the same time causes interference.
- In void setup(), always initalize all SPI devices as "off"
  - Note: $\overline{CS}$ is active LOW ("off" is HIGH)

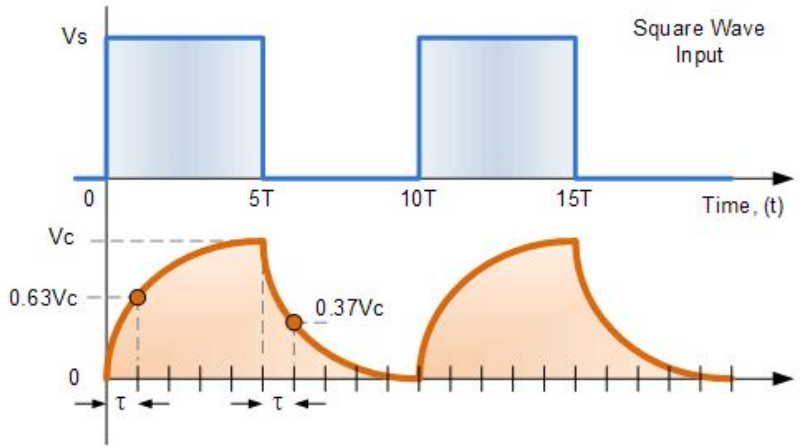
# $\mu$SD Card Module




1. SD cards are sensitive to interface to the pins
   - Keep wires short
   - Data lines need to be 3.3V, use level shifter is needed
2. FAT16 or FAT32 format
   - File naming - 8.3 (e.g., myfile12.csv)
3. Pinout
   - ◀ 5V - Power input(3.3V or 5V)
   - 3V output to power other devices
   - GND - Ground
   - ◀ CLK - Clock
   - ▶ DO - MISO
   - ◀ DI - MOSI
   - ◀ CS - Chip Select
   - CD - Card Detect

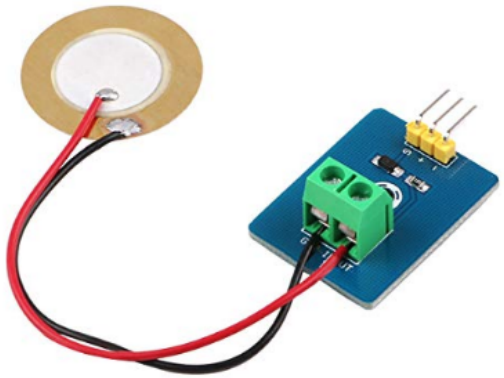
# RC Time Constant



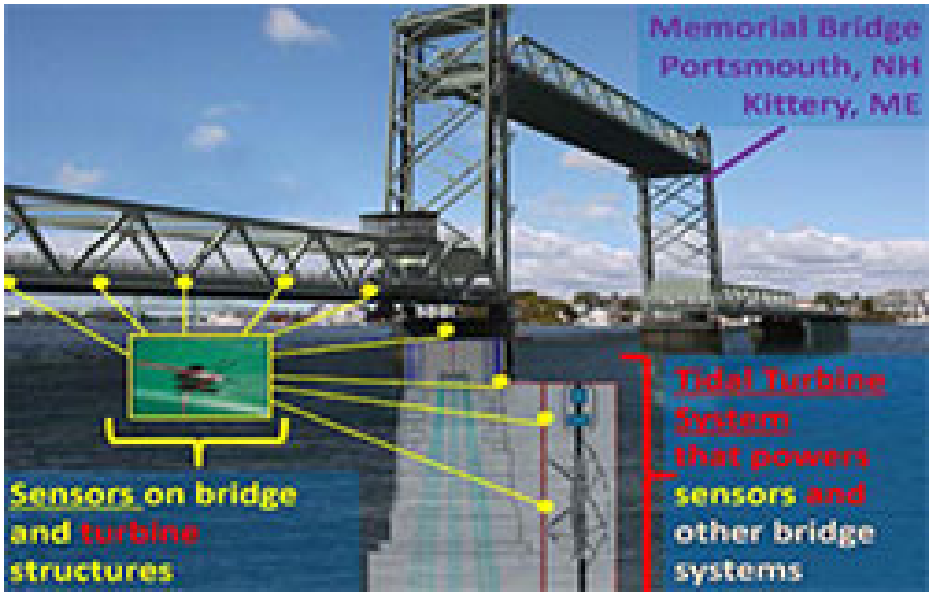

- Up until now, we have considered wires as ideal conductors; however:
  - Wires have non-zero resistance, longer and thinner wires have more resistance.
  - When two wires are close to each other, there is a parasitic capacitance between them.
- The time constant ($\tau = \frac{1}{RC}$) of an RC circuit determines the amount of time it takes a square wave input to reach 63% of it's final value.
- Some communication protocols expect sharp transitions between 0 and 3.3V for clock and data signals.

# Piezoelectric Elements - Revisited



- The piezoelectric effect is the appearance of electrical potential (voltage) across the side of a crystal when subject to mechanical stress.
- Conversely, a crystal becomes mechanically stressed (deformed in shape) when a voltage is applied across opposite faces.
- By utilizing an analogRead(), the vibration (change in mechanical stress) can be monitored over time.

# Structural Engineering Sensors



# FAT File System - SDCard Project (Argon Only)

| Feature | FAT32 | NTFS |
|---|---|---|
| Maximum Partition Size | 2TB | 2TB |
| Maximum File Size | 4GB | 16TB |
| Maximum File Name | 8.3 Characters | 255 Characters |
| File/Folder Encryption | No | Yes |
| Fault Tolerance | No | Auto Repair |
| Security | Network Only | Local and Network |
| Compression | No | Yes |
| Compatibility | Win 95/98/2000/XP and the derivations | Win NT/2000/XP/Vista/7 and the later versions |

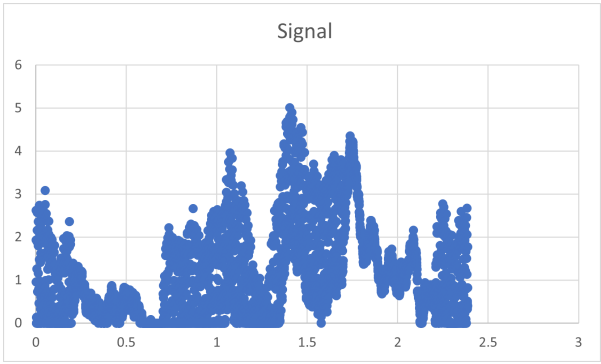
The FAT (File Allocation Table) file system, originally designed in 1977 for floppy disks, is simple and robust. It offers good performance in very light-weight implementations, but does not deliver performance, reliability and scalability afforded by modern file systems (such as NTFS or exFAT).

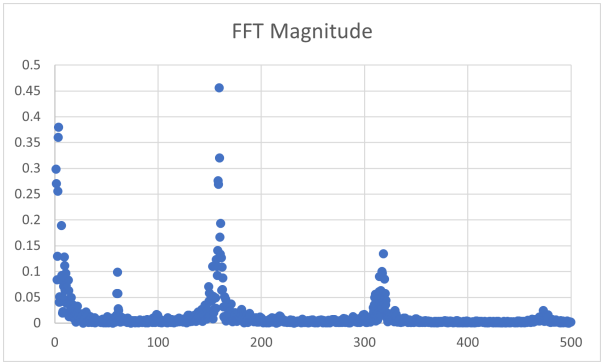
*Note: FAT file name follows a 8.3 format (e.g., ABCDEFGH.txt). We will be appending two digits, so our base name can be up to 6 characters (e.g. FILE_BASE_NAME = "mydata" → mydata42.csv).*

# Galaxy S9+ Vibration Analysis

Using the FFT Tutorial in Class_Materials repository

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | TimeStamp | Signal | Frequency | FFT Magnitude | Complex FFT |
| 2 | 0.00061 | 2.62 | 0.4209321 | 2.000009766 | 4096.02 |
| 3 | 0.00119 | 1.93 | 0.8418642 | 0.836477508 | -552.77950380473+1621.47055713326i |
| 4 | 0.00177 | 0.96 | 1.2627963 | 0.298364661 | -414.982558995766-448.522671528173i |
| 5 | 0.00235 | 0.13 | 1.6837284 | 0.270681692 | 353.443826952842-427.069259698417i |
| 6 | 0.00293 | 0 | 2.1046606 | 0.083873335 | -72.9068609990069+155.532672030055i |
| 7 | 0.003509 | 0 | 2.5255927 | 0.129856545 | 252.456289478309+83.62538763186061 |
| 8 | 0.004089 | 0 | 2.9465248 | 0.255636434 | -516.88842919695+83.210943362584i |
| 9 | 0.004669 | 0 | 3.3674569 | 0.360085774 | 423.28074046682-603.882664071708i |
| 10 | 0.005249 | 0.14 | 3.788389 | 0.379410535 | 88.80273300538+771.941714466294i |
| 11 | 0.005829 | 1.26 | 4.2093211 | 0.04066392 | -41.3210095359081-72.3054897410451i |
| 12 | 0.006409 | 2.16 | 4.6302532 | 0.051383144 | 94.6397062012862-46.0135075977235i |
| 13 | 0.006988 | 2.57 | 5.0511853 | 0.084507321 | 25.9352596801745-171.116717569232i |
| 14 | 0.007568 | 1.89 | 5.4721175 | 0.041174283 | 36.0724144460193-76.2199128809511i |
| 15 | 0.008148 | 0.73 | 5.8930496 | 0.04976769 | -73.9953870941929-70.0944447984849i |



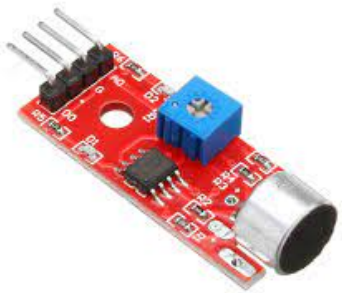

The Galaxy S9 vibrates at 159.11 Hz

# Assignment: L14_DataXfer (Argon Only)

- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L14_01_Vibration
   - Connect the piezo sensor, a button, and a $\mu$SD module to your Particle.
   - Each time button is pressed, execute a loop 4096 times:
     - Every 500$\mu sec$, collect piezoelectric data (without using a delay).
     - Save the piezo data and a timestamp (converting micros() to seconds) to a 2-dimensional array.
   - When the loop is complete, write the timestamp and data to a file.
   - Collect vibration data from the lathe, cell phone vibration, other machines at FUSE.
   - Use Excel and the FFT Tutorial (class_slides) to resample graph data in frequency domain (This process will be reviewed as a class.).

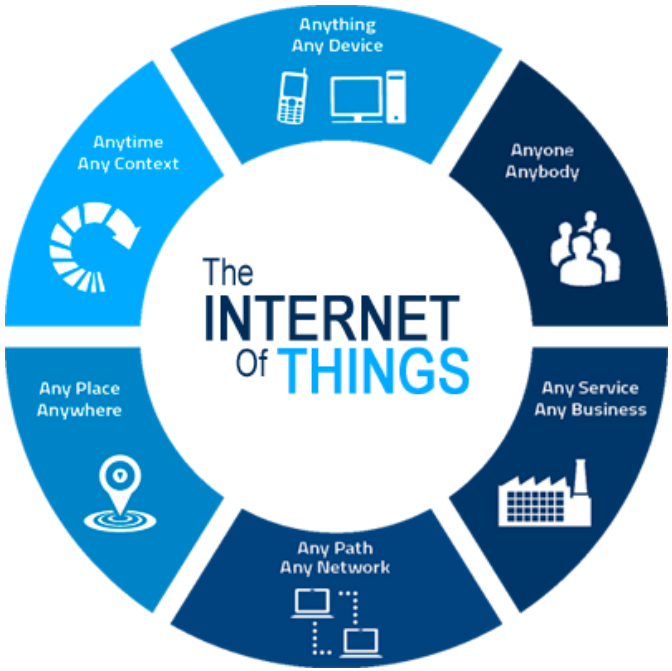
# Assignment: L14_DataXfer EXTRA



- Notebook: schematic
- Fritzing diagram
- Wire your circuit
- Write the code

1. L14_01_Vibration
   - Borrow a microphone.
   - Install in place of the piezo sensor.
   - Use Physics Toolbox Sensor Suite - Tone Generator to create a tone of a specific frequency.
   - Record/save with the Photon2, and create an FFT

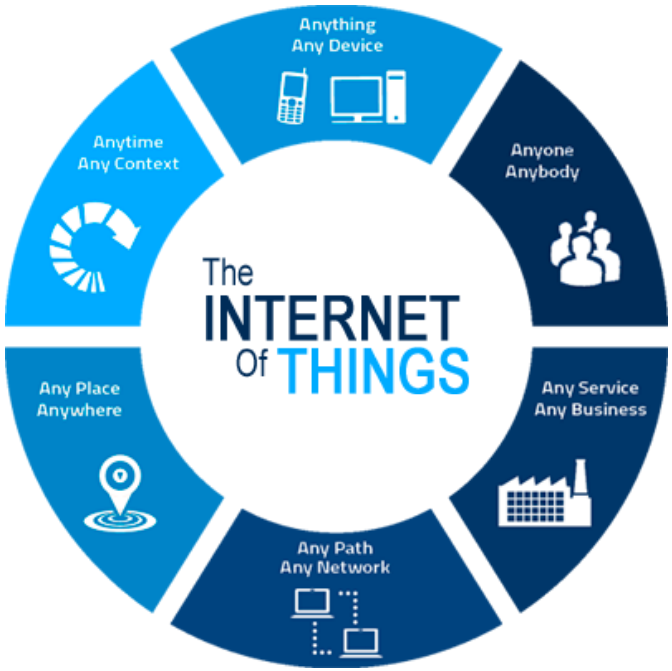
# Module 14 Review




- Learning Objectives
  1. Bluetooth
  2. GPS
  3. UART / SPI
  4. LoRa
- Additional Items
  1. 3D Modeling Lesson 6 - ESP32-CAM
  2. Quiz 10

# Module 15 - In the Wild

# Module 15 Objectives

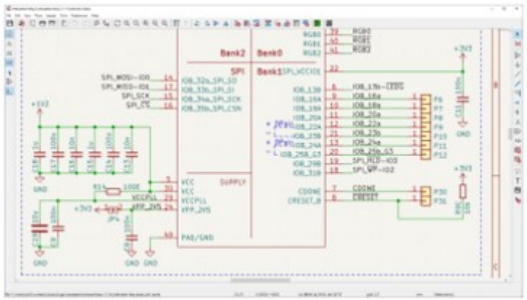


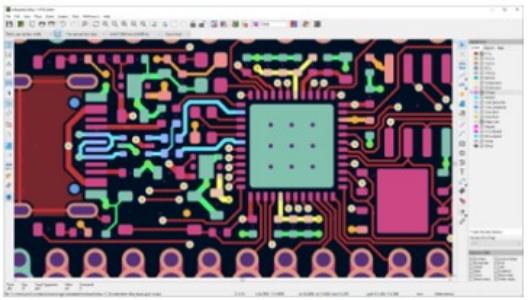

- Learning Objectives
  1. PCB Design
  2. Power and Reset
  3. Watchdogs
  4. Power Management
  5. Cloud Flash
  6. Hosting MQTT/NodeRed

# KiCad: Breadboard are good, but PCBs are better

## Schematic Capture

KiCad's Schematic Editor supports everything from the most basic schematic to a complex hierarchical design with hundreds of sheets. Create your own custom symbols or use some of the thousands found in the official KiCad library. Verify your design with integrated SPICE simulator and electrical rules checker.



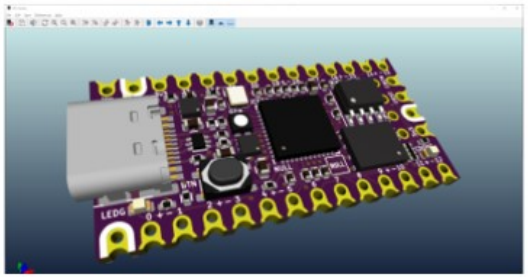


## PCB Layout

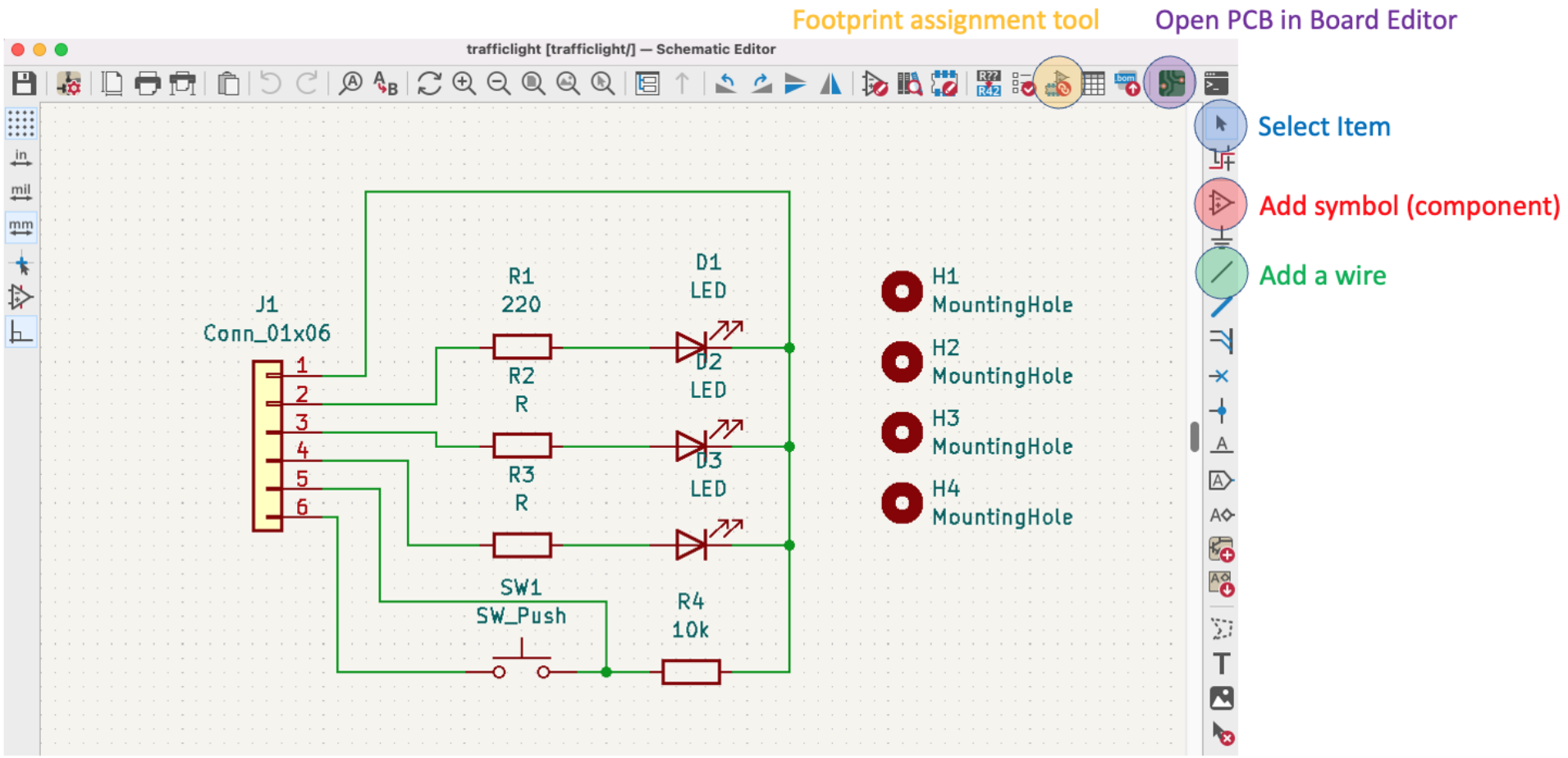
KiCad's PCB Editor is approachable enough to make your first PCB design easy, and powerful enough for complex modern designs. A powerful interactive router and improved visualization and selection tools make layout tasks easier than ever.

## 3D Viewer

KiCad's 3D Viewer allows easy inspection of your PCB to check mechanical fit and to preview your finished product. A built-in raytracer with customizable lighting can create realistic images to show off your work.

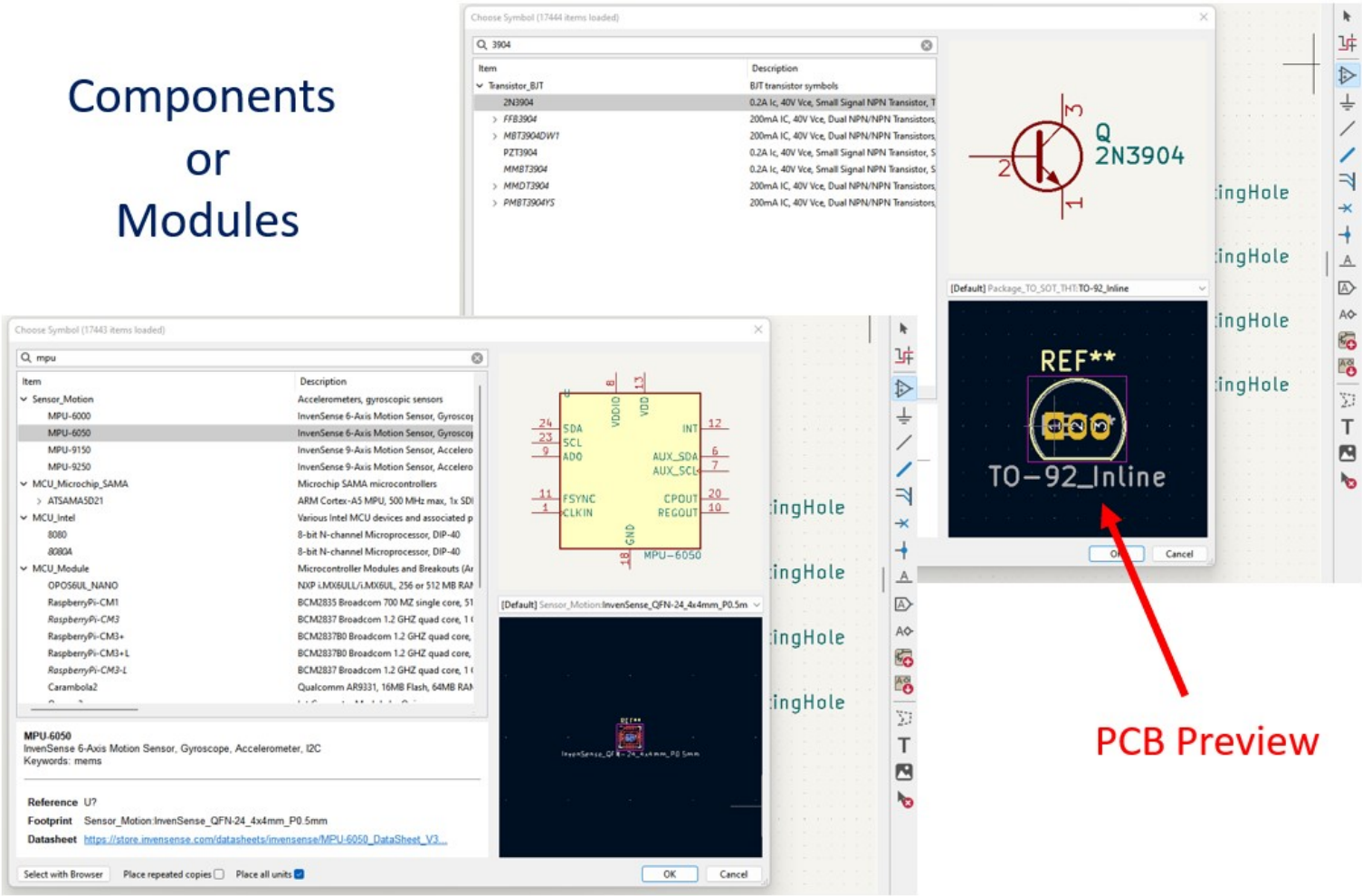


# Start with schematics

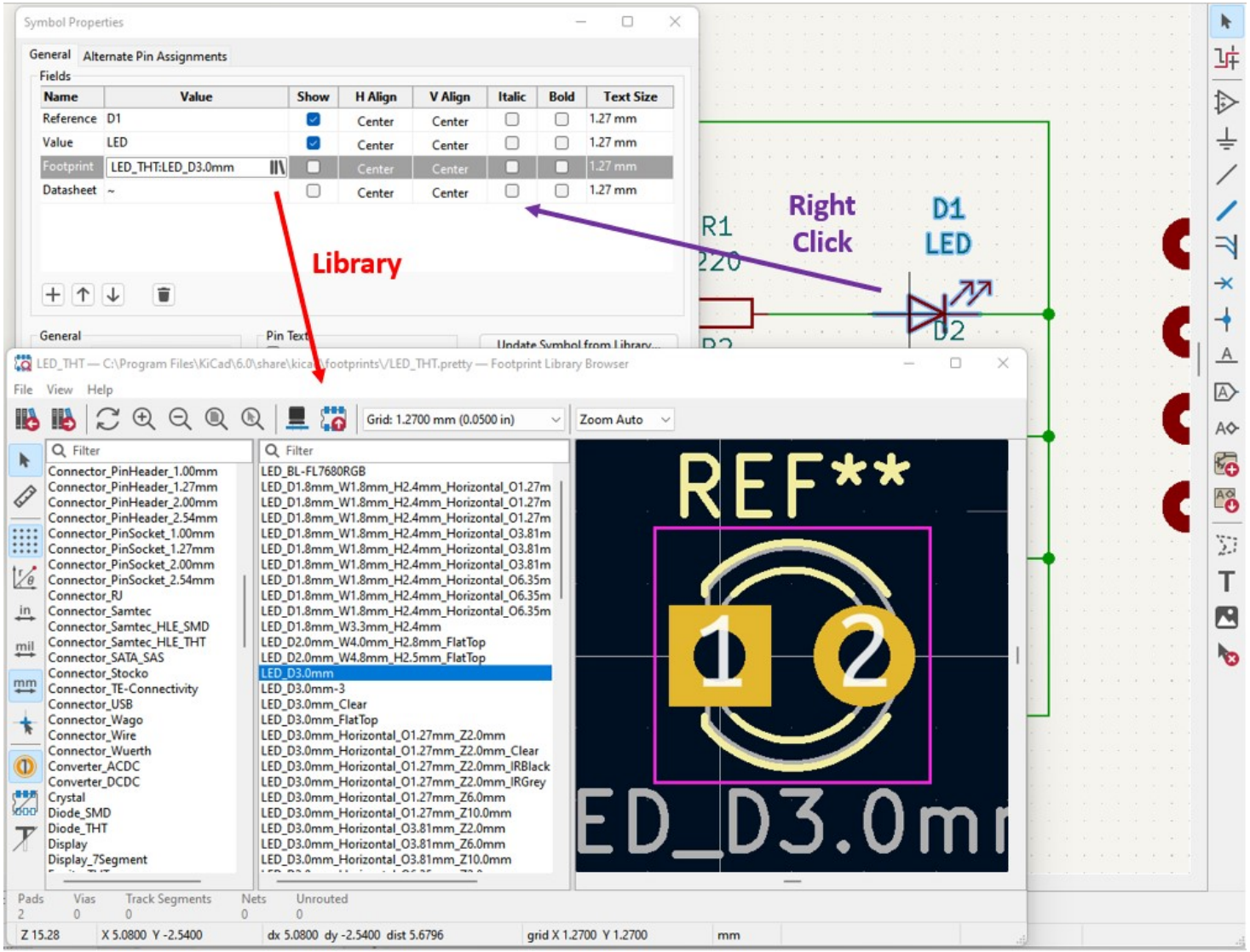


# Rich library of components

Components
or
Modules



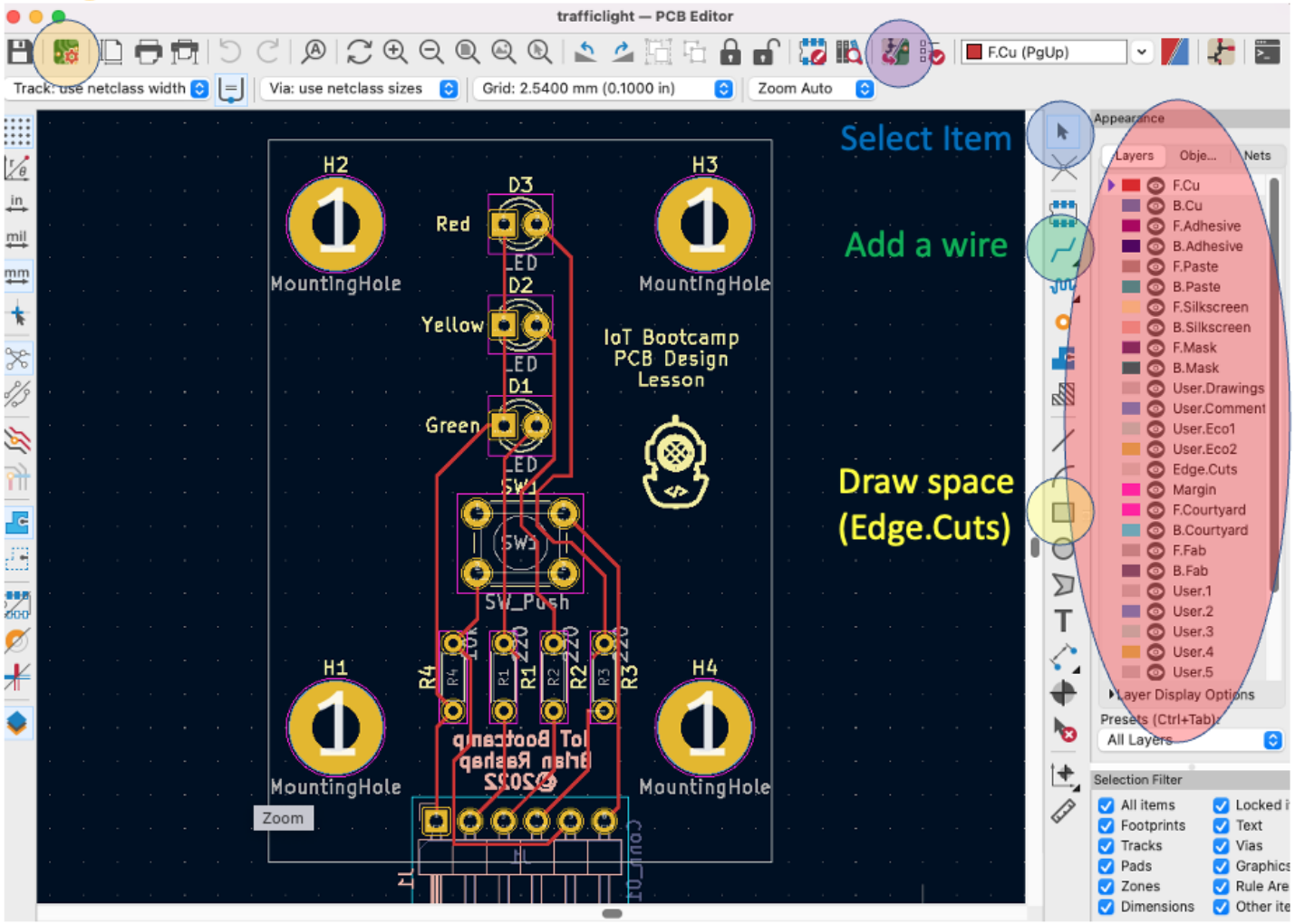
# Select a footprint for each component



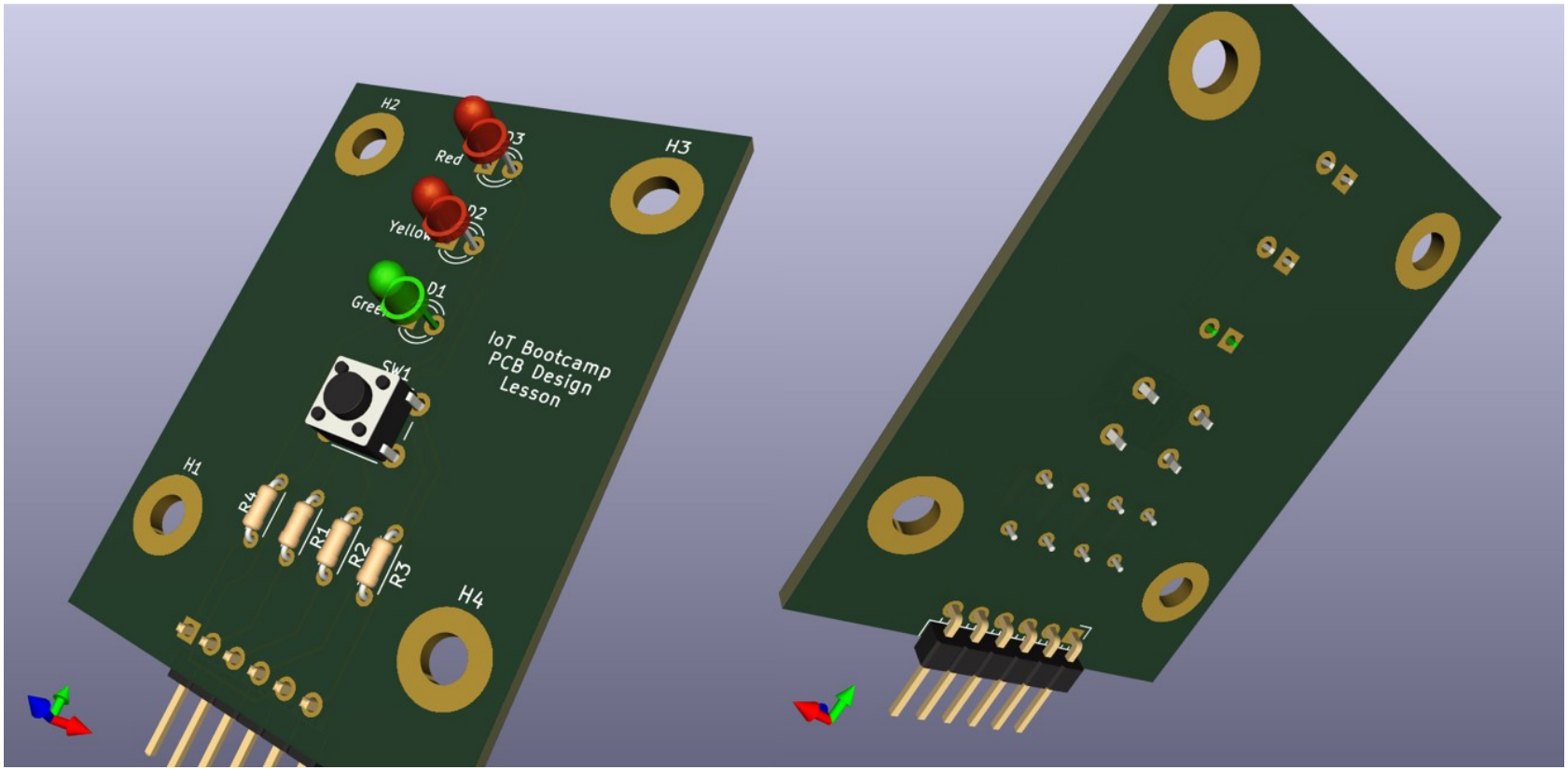
# Generate PCB, Route Wires

Edit board setup including layers, design rules, and various defaults

Update PCB with changes make to schematic



# 3D View


H2
H3
H1
H4
Red
Yellow
D3
D2
D1
SW1
R4
R1
R2
R3
IoT Bootcamp
PCB Design
Lesson

# Import into KiCad

### Import Symbols

Using the KiCad (*.lib) file:

1. In KiCad, go to **Tools > Edit Schematic Symbols**.
2. Click on **Preferences > Manage Symbol Libraries**.
3. On the **Global Libraries** tab, click on **Browse Libraries** (the ***small folder icon*** below) and select the .lib file. Then click **Open**. The library will appear, click OK.
4. Toggle the search tree on, and navigate to the symbol you imported. Double-clic over it to open the file.

### Import Footprints

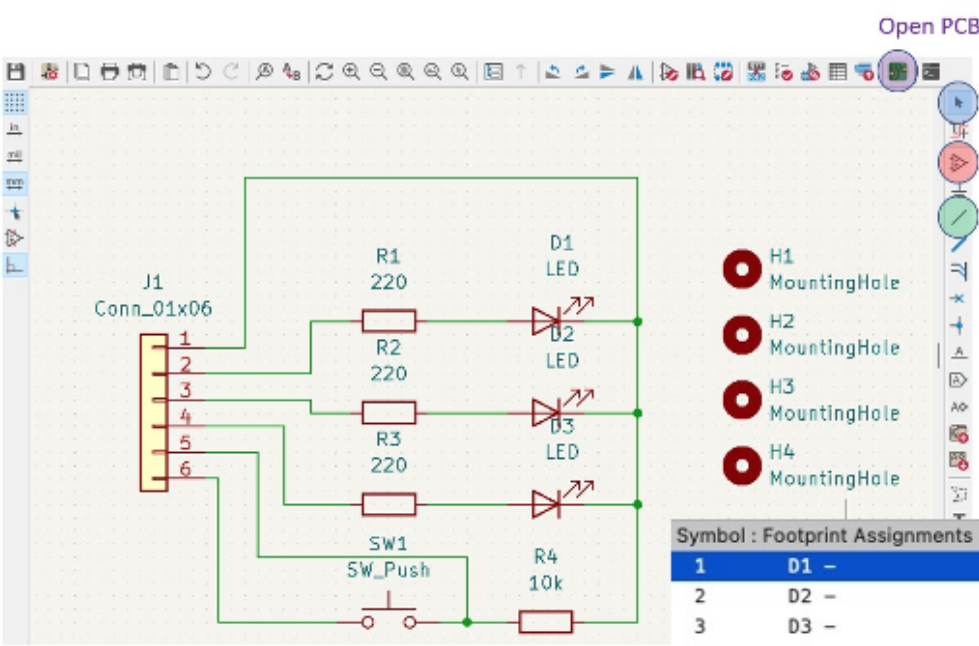
Using the *.kicad_*mod* file:

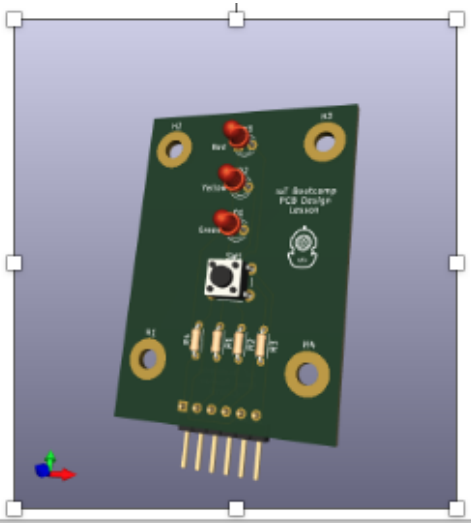
1. In KiCad, go to Tools > Edit PCB Footprints.
2. Click on Preferences > Manage Footprint Libraries.
3. On the **Global Libraries** tab, click on **Browse Libraries** (the ***small folder icon*** below) and navigate to the **Folder** of the downloaded .kicad_mod file. Then click **Open,** and the library will appear. If the path doesn't have the same name. you can rename it as the part.
4. In the table, make sure that the Plugin Type is set to **KiCad**. Then click **OK**.
5. Toggle the search tree on, and navigate to the footprint you imported. Double-click over it to open the file.

Using the **mod* file:

1. Follow the same steps above from step 1 to step 3.
2. In the table, make sure that the Plugin Type is set to **Legacy**. Then click **OK**.
3. Toggle the search tree on, and navigate to the footprint you imported. Double-click over it to open the file.

# Assignment: L15_TrafficPCB

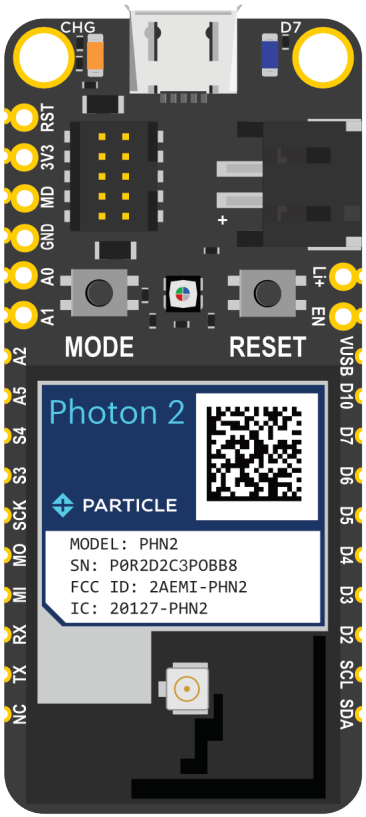


Symbol : Footprint Assignments

| | | | |
|---|---|---|---|
| 1 | D1 – | LED | : LED_THT:LED_D3.0mm |
| 2 | D2 – | LED | : LED_THT:LED_D3.0mm |
| 3 | D3 – | LED | : LED_THT:LED_D3.0mm |
| 4 | H1 – | MountingHole | : MountingHole:MountingHole_3.5mm_Pad |
| 5 | H2 – | MountingHole | : MountingHole:MountingHole_3.5mm_Pad |
| 6 | H3 – | MountingHole | : MountingHole:MountingHole_3.5mm_Pad |
| 7 | H4 – | MountingHole | : MountingHole:MountingHole_3.5mm_Pad |
| 8 | J1 – | Conn_01x06 | : Connector_PinHeader_2.54mm:PinHeader_1x06_P2.54mm_Horizontal |
| 9 | R1 – | 220 | : Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P5.08mm_Horizontal |
| 10 | R2 – | R | : Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P5.08mm_Horizontal |
| 11 | R3 – | R | : Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P5.08mm_Horizontal |
| 12 | R4 – | 10k | : Resistor_THT:R_Axial_DIN0204_L3.6mm_D1.6mm_P5.08mm_Horizontal |
| 13 | SW1 – | SW_Push | : Button_Switch_THT:SW_PUSH_6mm_H5mm |

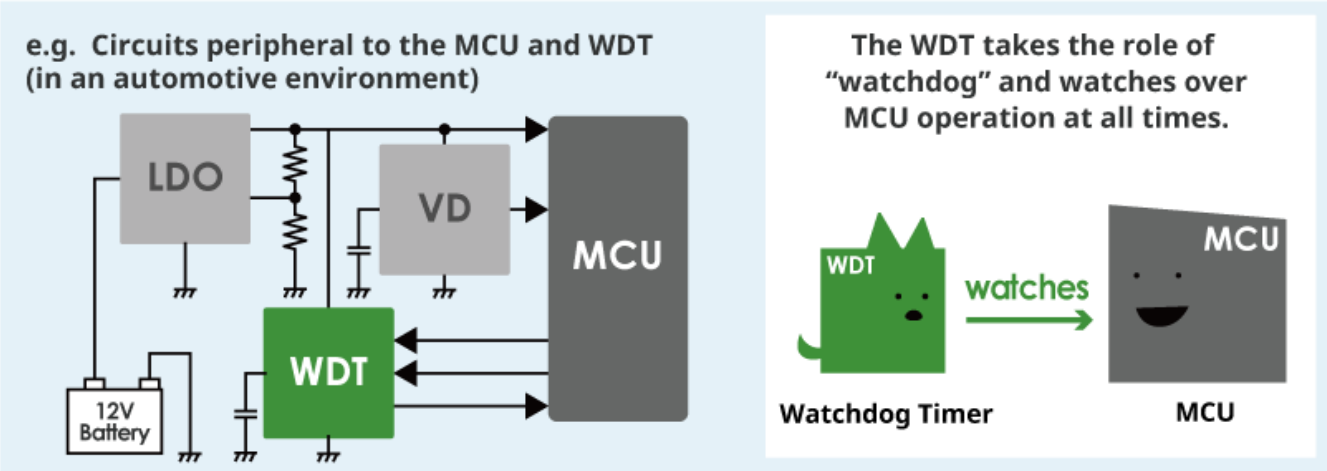
# EN and RST pins




The Photon2 has two pins that are extremely useful in real world situations:

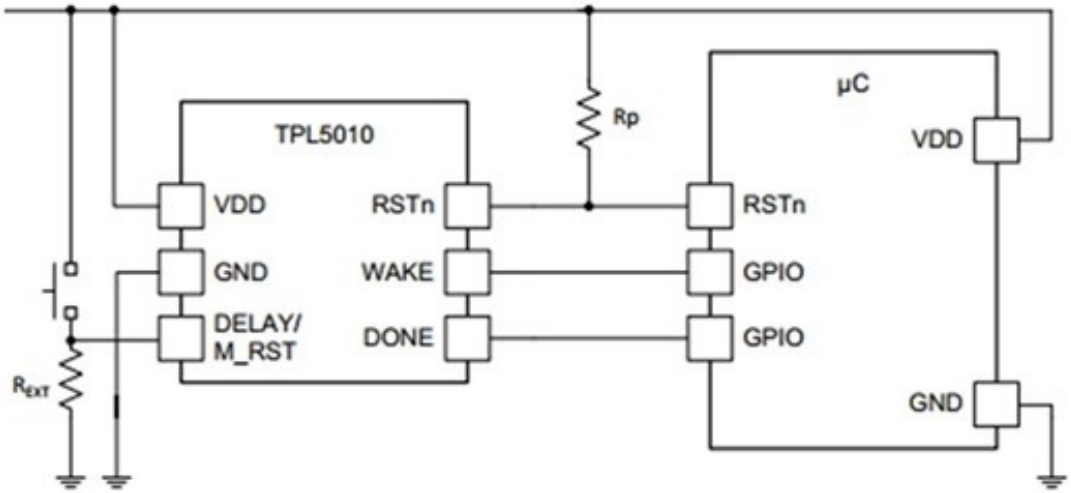
1. EN pin
   - The EN pin is not a power pin, per se, but it controls the 3V3 power.
   - Device enable pin is internally pulled-up. To disable the device (force the device into a deep power-down state), connect this pin to GND.
   - This pin is essentially an on/off pin.
2. RST pin
   - Active-low system reset input. This pin is internally pulled-up.

# Watchdog Timer




The watchdog timer communicates with the MCU at a set interval. If the MCU does not output a signal, outputs too many signals or outputs signals that differ from a predetermined pattern, the timer determines that the MCU is malfunctioning and sends a reset signal to the MCU.

# Hardware Watchdog Timers - TPL5010




The timeout frequency is set by resistor $R_{EXT}$ using a formula from the data sheet.

| Timeout Interval | Calculated Resistance |
|---|---|
| 1 minute | 22 Ω |
| 5 minutes | 43 Ω |
| 30 minutes | 92 Ω |
| 1 hour | 125 Ω |
| 2 hours | 170 Ω |

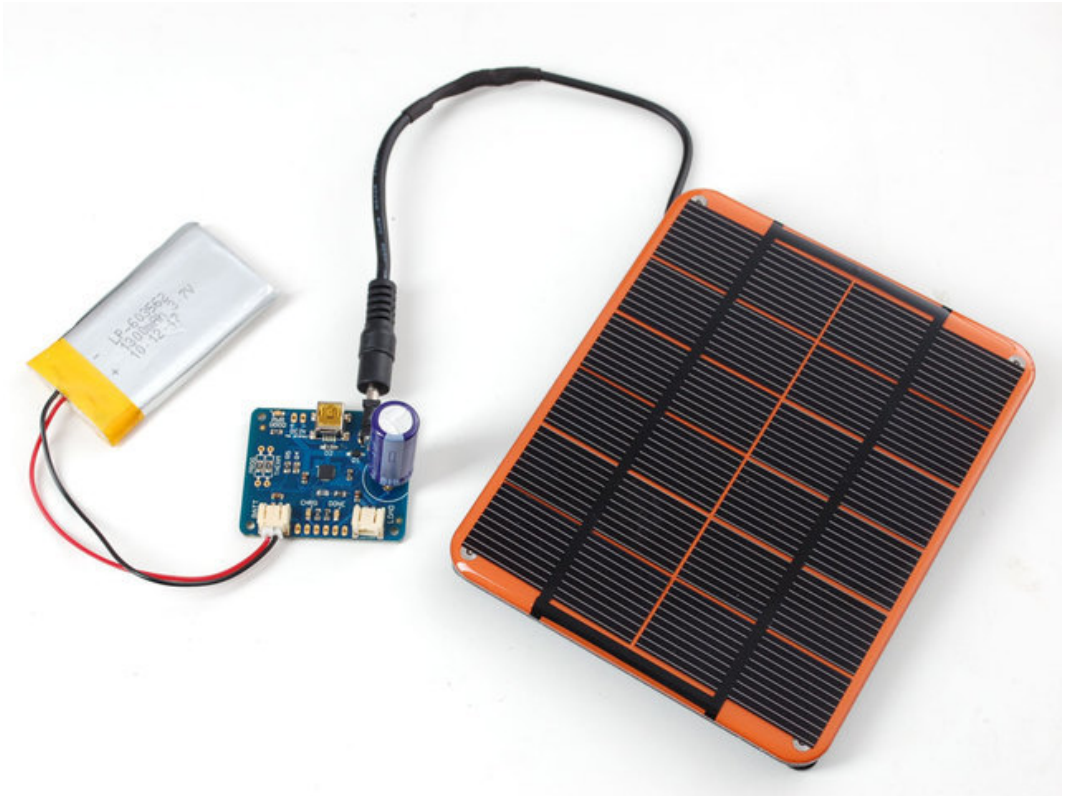
# Application Watchdog

The Photon2 also has a watchdog timer that can be implemented in code. It is not as robust as the hardware timer, but better than nothing.

```
1  // Prototype
2  // ApplicationWatchdog(unsigned timeout_ms, std::function<void(void)> fn, unsigned
       stack_size=DEFAULT_STACK_SIZE);
3
4  // Global variable to hold the watchdog object pointer
5  ApplicationWatchdog *wd;
6
7  void watchdogHandler() {
8    // Do as little as possible in this function, preferably just a reset
9    System.reset(RESET_NO_WAIT);
10 }
11
12 void setup() {
13   // Start watchdog. Reset the system after 60 seconds if the application is unresponsive
14   // The stack_size default is 512, but this is too small. Use at least 1536.
15   wd = new ApplicationWatchdog(60000, watchdogHandler, 1536);
16 }
17
18 void loop() {
19   while (some_long_process_within_loop) {
20     ApplicationWatchdog::checkin(); // resets the AWDT count
21   }
22 }
23 // AWDT count reset automatically after loop() ends
```

# Solar Charging

Should be easy, but isn't.



# Power Management

System.sleep() can be used to lower power consumption and increase battery life when the Particle is only needed intermittently to interact with the environment.

| | STOP | ULTRA_LOW_POWER | HIBERNATE |
|---|---|---|---|
| Relative power consumption | Low | Lower | Lowest |
| Relative wake options | Most | Some | Fewest |
| Execution continues with variables intact | ✓ | ✓ | |

| Device | GPIO | RTC | Analog | Serial | BLE | Network |
|---|---|---|---|---|---|---|
| T523 Eval | 139 uA | 139 uA | 140 uA | 564 uA | 214 uA | 21.7 mA |
| T402 Eval | 114 uA | 114 uA | 117 uA | 530 uA | 186 uA | 16.9 mA |
| Boron 2G/3G | 171 uA | 174 uA | 178 uA | 610 uA | 494 uA | 16.4 mA |
| Boron LTE | 127 uA | 128 uA | 130 uA | 584 uA | 442 uA | 14.2 mA |
| B402 SoM | 48 uA | 47 uA | 48 uA | 557 uA | 130 uA | 9.5 mA |
| B523 SoM | 54 uA | 55 uA | 56 uA | 537 uA | 139 uA | 22.8 mA |
| P2 | 579 uA | 572 uA | | | | |
| Argon | 82 uA | 81 uA | 82 uA | 520 uA | 141 uA | 21.3 mA |
| Electron | 2.42 mA | 2.55 mA | n/a | n/a | n/a | n/a |
| Photon | 2.76 mA | 2.83 mA | n/a | n/a | n/a | n/a |

# Ultra Low Power example

The below code places the Photon2 in Ultra_Low_Power mode, enabling it to be awakened either by a RISING signal on Pin D0 or after the time specified by the variable sleepDuration.

```
void sleepULP() {
  SystemSleepConfiguration config;
  config.mode(SystemSleepMode::ULTRA_LOW_POWER).gpio(D0,RISING).duration(sleepDuration);
  SystemSleepResult result = System.sleep(config);
  delay(1000);
  if (result.wakeupReason() == SystemSleepWakeupReason::BY_GPIO) {
    Serial.printf("Awakened by GPIO %i\n",result.wakeupPin());
  }
  if (result.wakeupReason() == SystemSleepWakeupReason::BY_RTC) {
    Serial.printf("Awakened by RTC\n");
  }
}
/*
 * Other wake-up modes (These modes DO NOT work with the Photon2)
 *  config.mode(SystemSleepMode::ULTRA_LOW_POWER).analog(pin, value, mode);
 *    mode can be: ABOVE, BELOW, or CROSS
 *  config.mode(SystemSleepMode::ULTRA_LOW_POWER).usart(Serial1);
 *    note: Serial can not be used for wake-up
 *  config.mode(SystemSleepMode::ULTRA_LOW_POWER).ble();
 *  config.mode(SystemSleepMode::ULTRA_LOW_POWER).network(NETWORK_INTERFACE_WIFI_STA);
 */
```

The Photon2 only has 2 options for wake: GPIO and RTC.

# Reminder: Not Everything is About Voltage

Recall $P = V * I$, it is possible to be current limited.

| Parameter | Symbol | Min | Typ | Peak | Unit |
|---|---|---|---|---|---|
| Operating Current (uC on, peripherals and radio disabled) | $I_{idle}$ | 21.4 | 23.2 | 23.8 | mA |
| Operating Current (uC on, BLE advertising) | $I_{ble\_adv}$ | 54.7 | 58.7 | 70.7 | mA |
| Operating Current (uC on, radio connected to access point) | $I_{wifi\_conn\_ap}$ | 54.6 | 60.5 | 265 | mA |

The $3.3V$ regulator can supply $500mA$. Typical power consumption:

- The Photon2 connected to Wifi can draw up to $265mA$.
- Each GPIO used as an output can supply $4 - 12mA$.
- Each NeoPixel uses up to $60mA$ at full brightness.
- Seeed sensors: dust - $90mA$, AQ sensor - $60mA$
- Plant Watering Pump: $130mA$

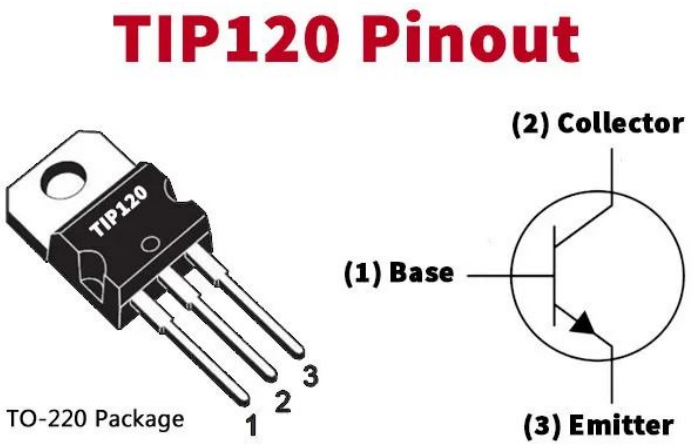
$V_{USB}$ can supply[3] up to $1500mA$[4]. External supply can add more current.

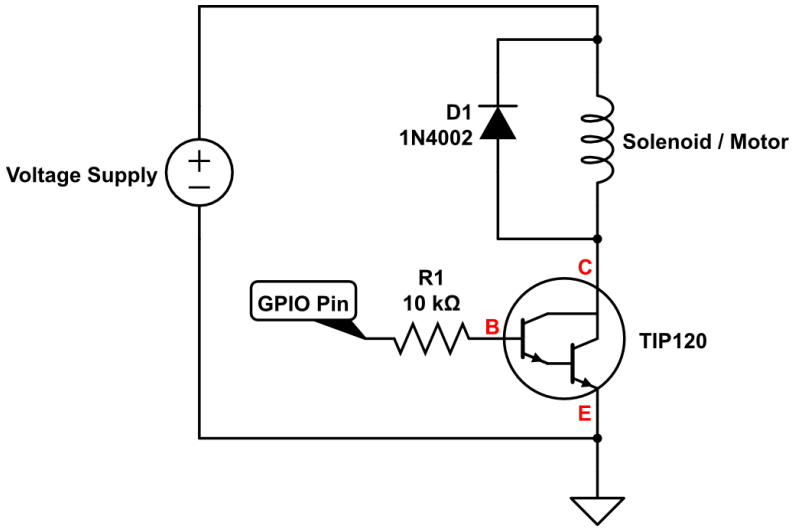
[3] It is all about power, so $1500mA$ at $5V$ is $2250mA$ at $3.3V$

[4] Specific amount depends on type of USB port used (2.0 vs 3.x vs USB-C)

# Assignment L15_01_HelloSolenoid

Similar to the Darlington Array used by the Stepper Motor, an individual Darlington transistor can be used to amplify current for a solenoid, motor, etc.



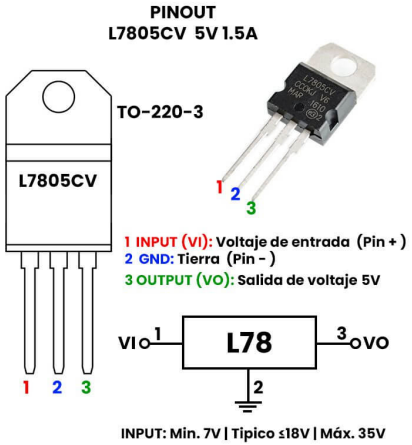



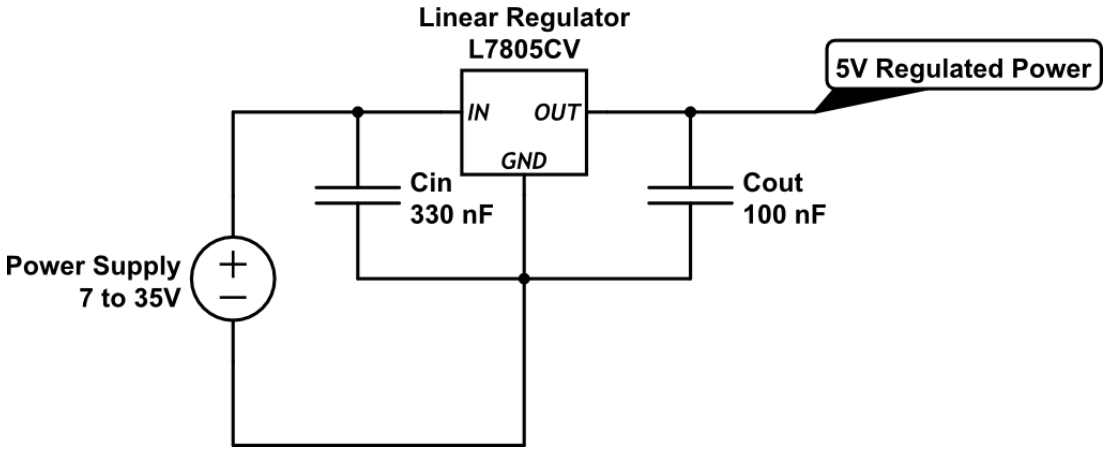

Assignment:

- Create a fritzing diagram of the above circuit.
- Borrow a solenoid from instructor and the Wire the circuit
- Use HelloWorld code to actuate the solenoid

# Linear Voltage Regulator

The L7805CV is a three-terminal positive voltage regulator that provides a fixed output voltage of 5 volts. It is commonly used in electronic circuits to ensure a stable power supply and protect sensitive components from voltage fluctuations. This provides a maximum of $1.5A$.



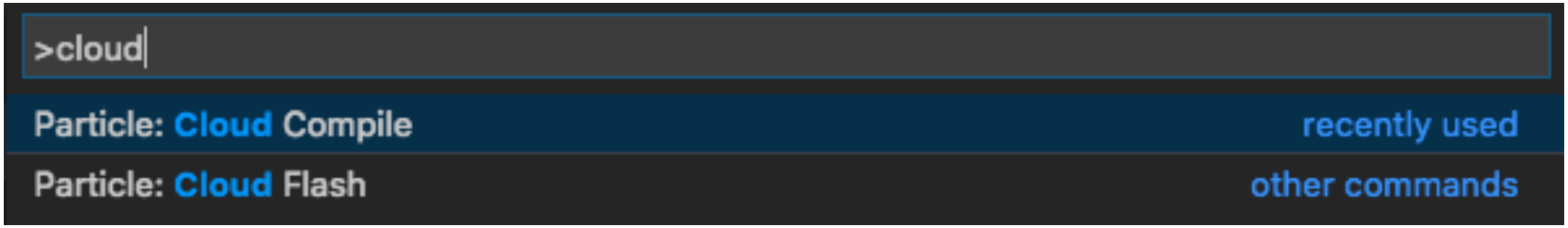




Alternatives:

- LM317 - adjustable ($1.2V$ to $37V$) voltage regulator
- AMS1117 - $3.3V$ voltage regulator

# Cloud Flash

During the deployment phase, it isn't convenient to have to hook up a USB cable to push updates. The Particle ecosystem allows for sending code over-the-air (OTA).

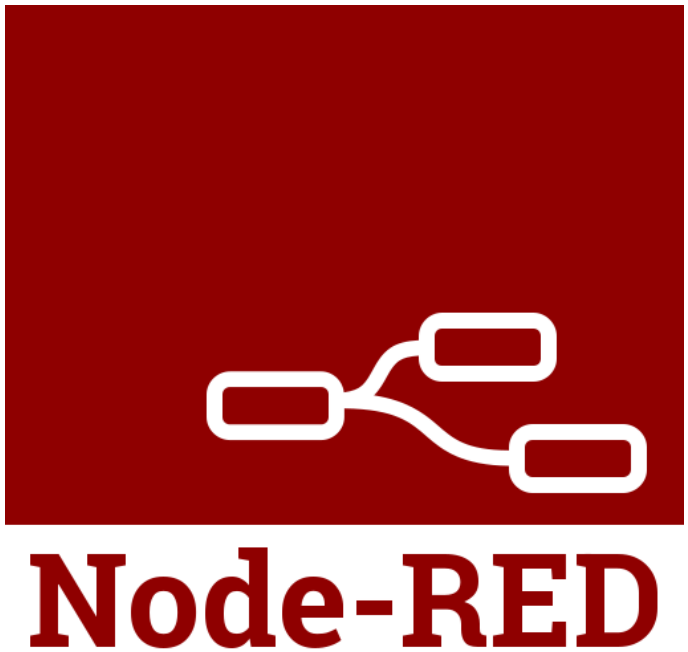



- Cloud Compile - compile your program and download the binary
- Cloud Flash - compile and flash it to the selected device OTA

The OTA operations require:

- The device is connected to the Particle Cloud (breathing cyan)
- The computer is into the same account that claimed the device.
- The DeviceID is set to the device name.

# Local MQTT Server

# Install Mosquitto (Windows - PowerShell)

```
// On Windows go to mosquitto.org/download

// Open three PowerShell windows and set each of them to
cd c:\'Program Files'\mosquitto

// PowerShell window #1: Start Mosquitto
.\mosquitto

// PowerShell window #2: Subscribe to the test topic (feed)
.\mosquitto_sub -h localhost -t feeds/test

// PowerShell window #3: Publish "Hello World"
.\mosquitto_pub -h localhost -t feeds/test -m "Hello World"
```

# Install Mosquitto (Windows - Git Bash)

```
// On Windows go to mosquitto.org/download

// Open three PowerShell windows and set each of them to
cd /c/'Program Files'/mosquitto

// PowerShell window #1: Start Mosquitto
./mosquitto

// PowerShell window #2: Subscribe to the test topic (feed)
./mosquitto_sub -h localhost -t feeds/test

// PowerShell window #3: Publish "Hello World"
./mosquitto_pub -h localhost -t feeds/test -m "Hello World"
```

# Install Mosquitto (Mac / Linux)

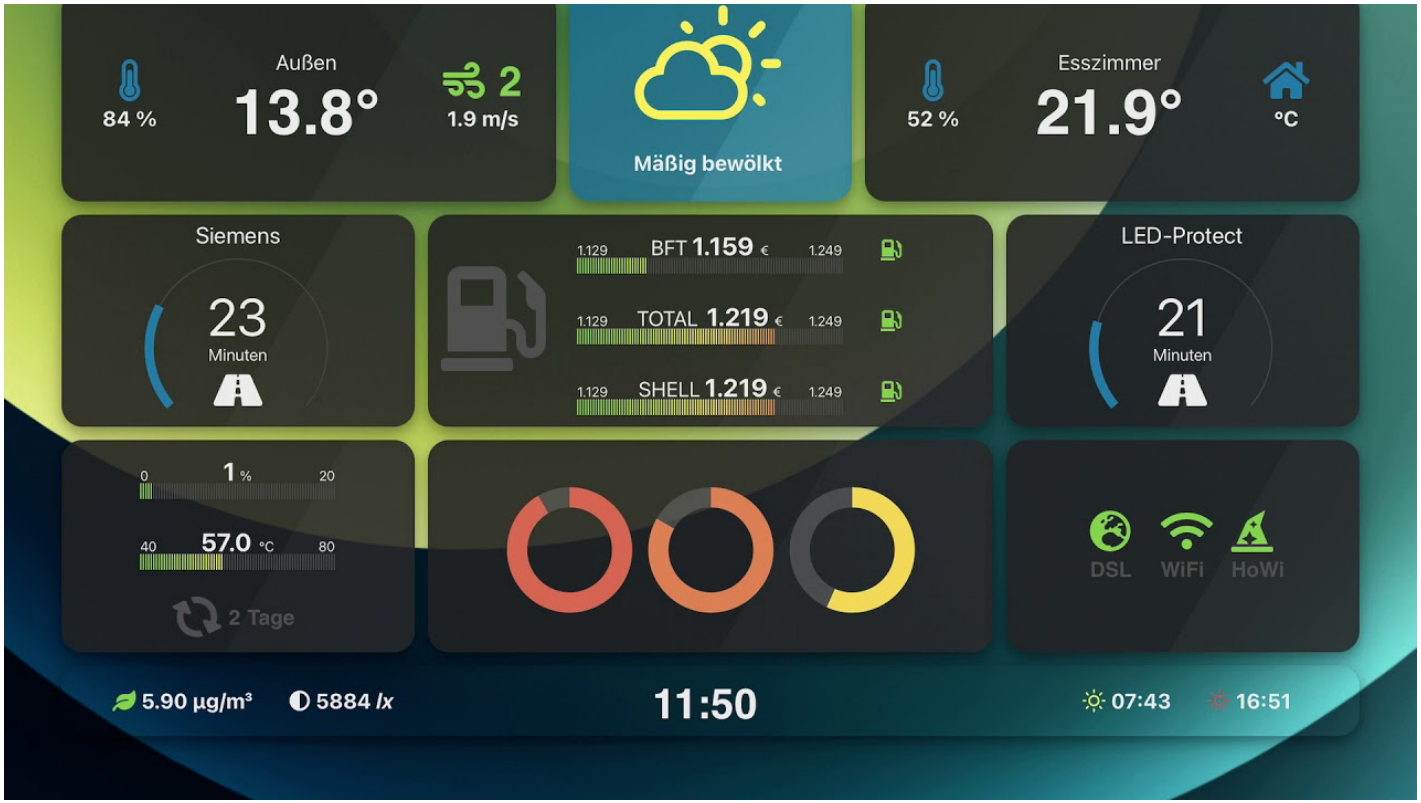
```
// On Windows go to mosquitto.org/download

// On Mac using Homebrew (Intel Processor)
brew install mosquitto

// On Mac using Homebrew (Apple M1/2/3 Processor)
arch -arm64 brew install mosquitto

// On Linux (Debian and Ubuntu)
sudo apt-get install mosquitto mosquitto-clients

// Open three terminals:

// Terminal #1: Start Mosquitto
sudo /usr/local/sbin/mosquitto -c /usr/local/etc/mosquitto/mosquitto.conf

// Terminal #2: Subscribe to the test topic (feed)
mosquitto_sub -h localhost -t feeds/test

// Terminal #3: Publish "Hello World"
mosquitto_pub -h localhost -t feeds/test -m "Hello World"
```

# Install Node-Red (Basic)

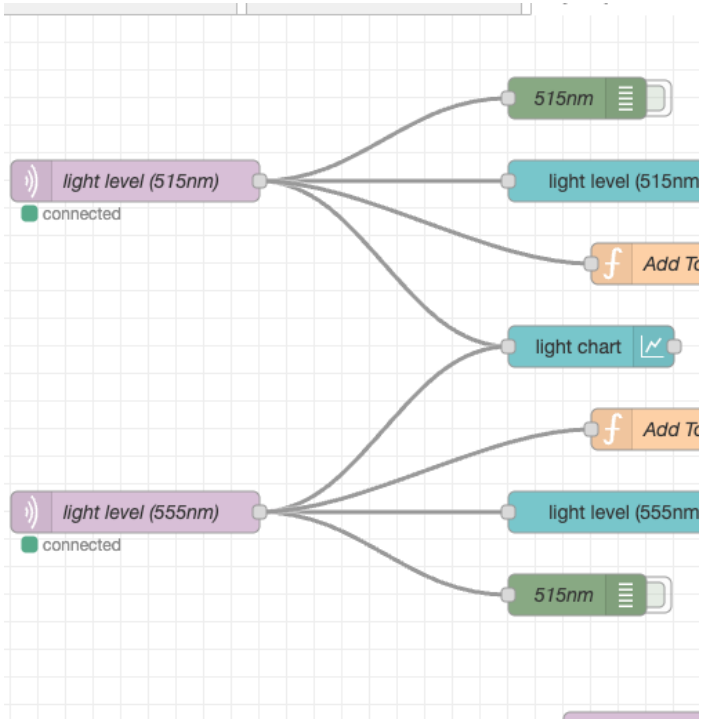
```
// Start With Installing nodejs:

// On Windows go to nodejs.org/en/

// On Mac using Homebrew (Intel Processor)
brew install node

// On Mac using Homebrew (Apple M1/2/3 Processor)
arch -arm64 brew install node

// On Linux (Debian and Ubuntu)
sudo apt install nodejs npm

// VERFIY: In terminal or PowerShell, verify using
node -v
npm -v

// On Windows (Powershell):
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy Unrestricted

// Install Node-RED (all platforms)
npm install -g --unsafe-perm node-red

// Start node-red
node-red
```

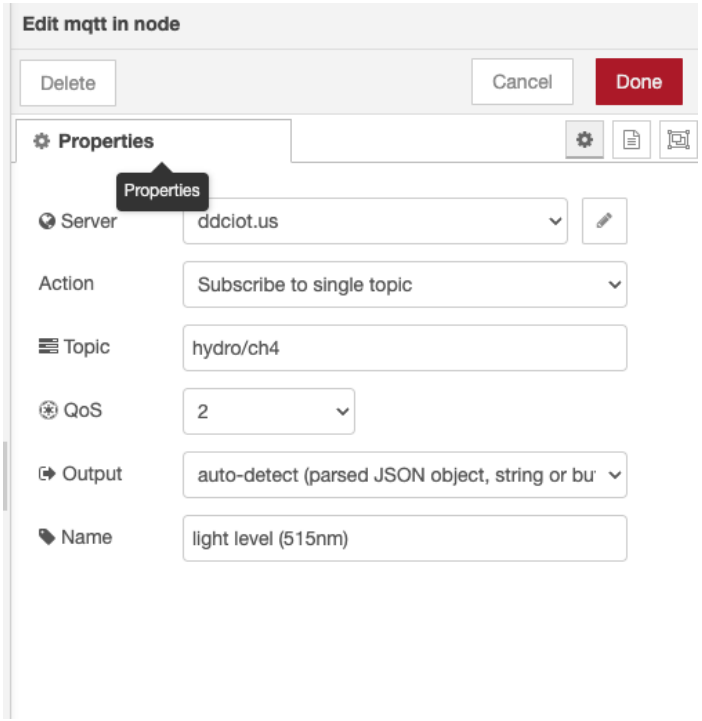
In a browser URL bar, go to localhost:1880

# Node Red Dashbaard

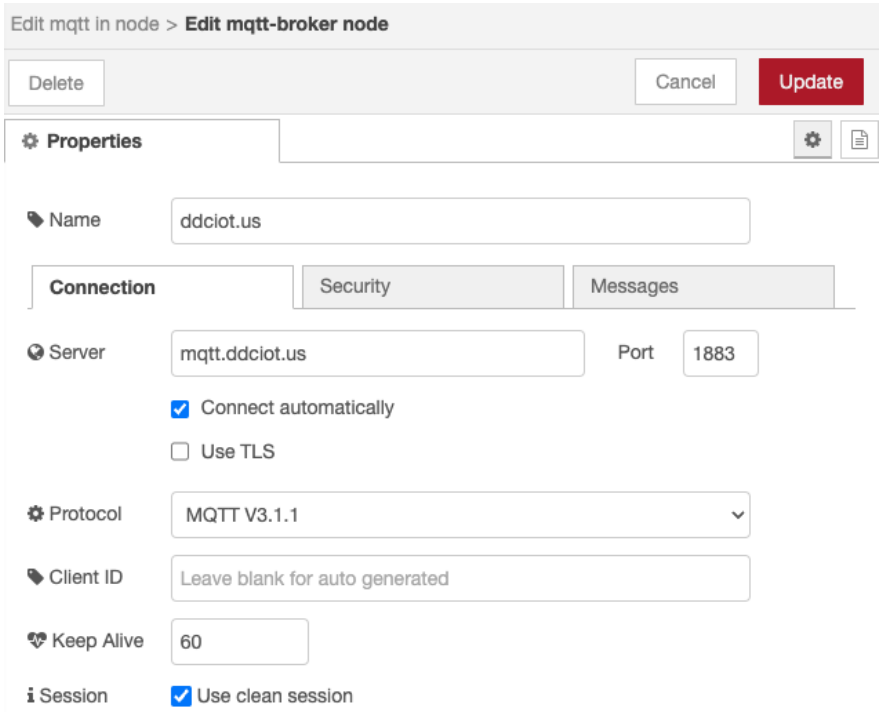


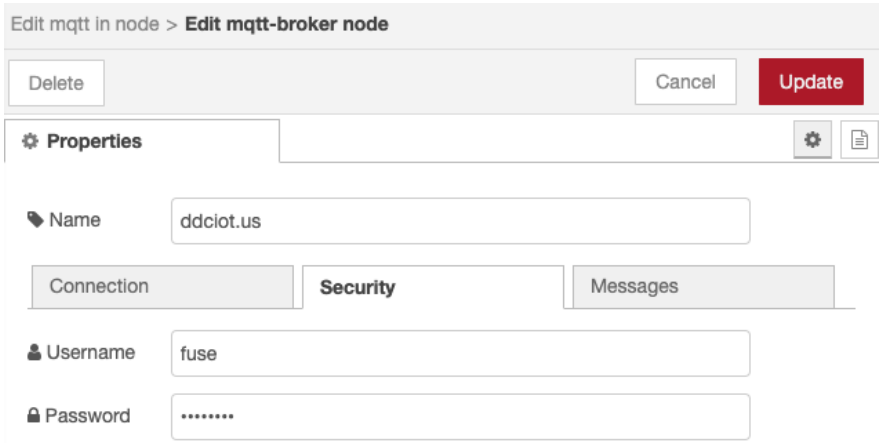
# Node Red MQTT Node



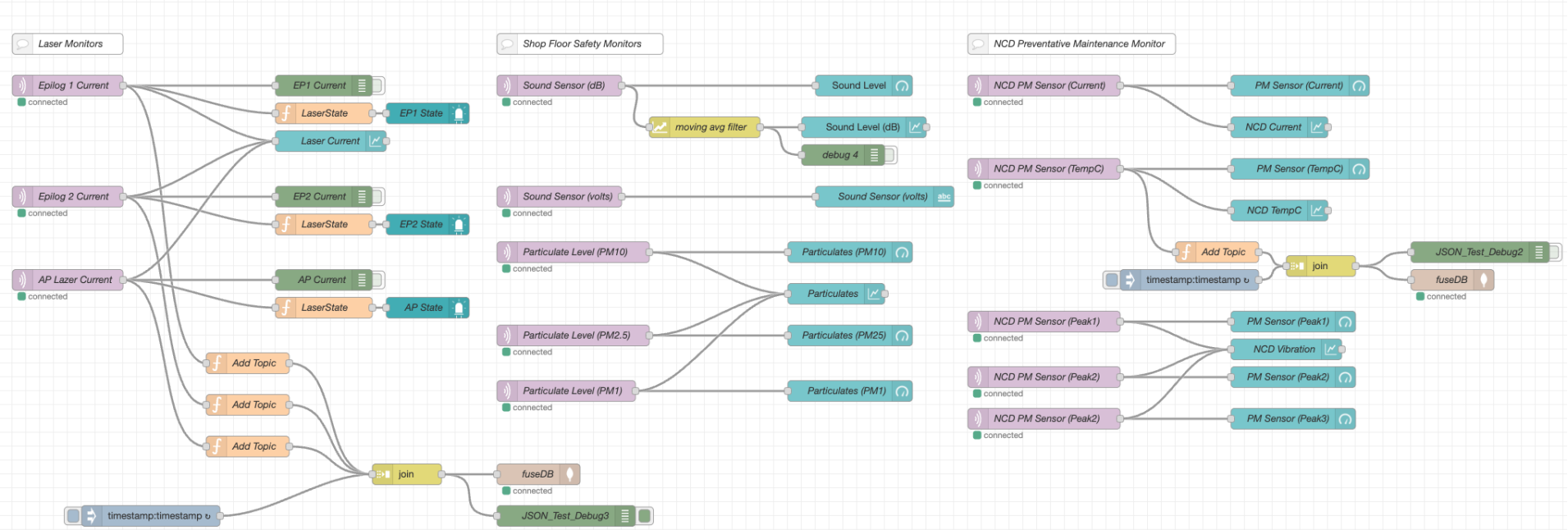



# Node Red MQTT Connection

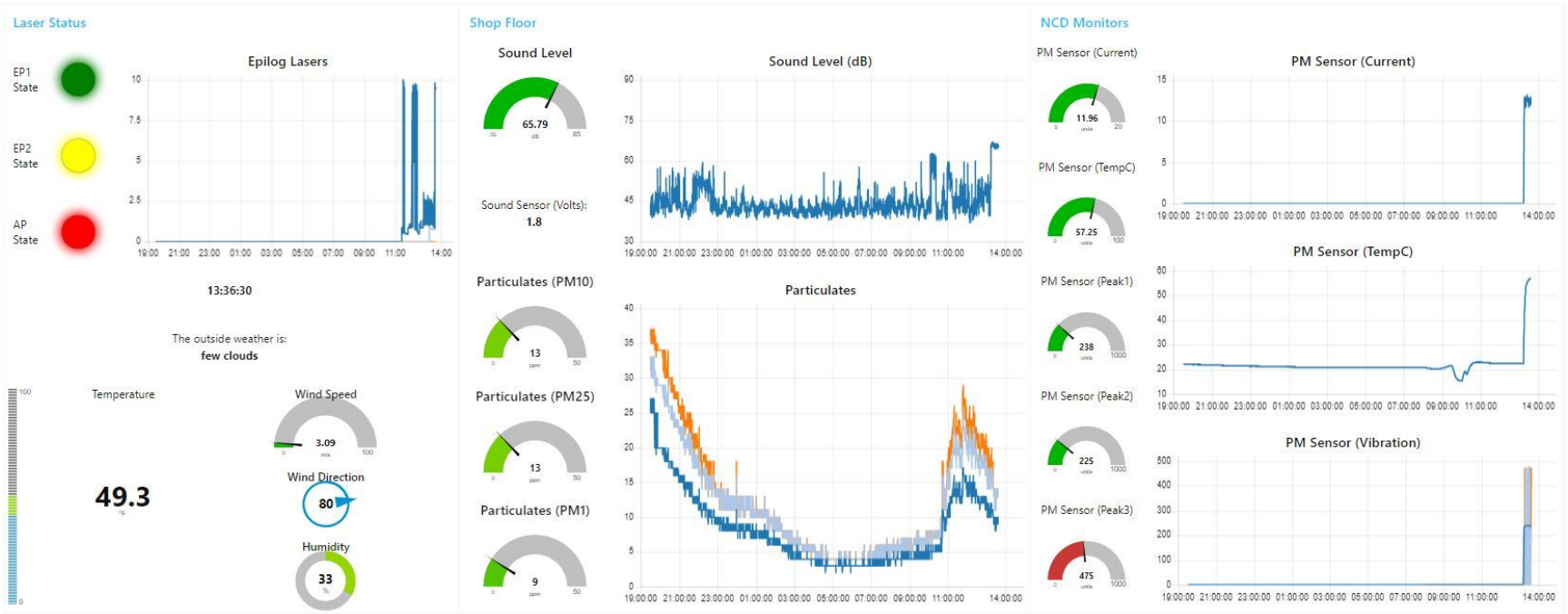


# Node Red Smart Fuse Flow



# Node Red Smart Fuse Dashboard



# Install Mosquitto MQTT Broker (Detailed: Linux)

## Install Mosquitto

```
// install Mosquitto broker and client
sudo apt-get install mosquitto mosquitto-clients
// start the broker and enable at startup
sudo systemctl enable mosquitto.service
sudo systemctl start mosquitto.service
// open new terminal window
mosquitto_sub -h localhost -t feeds/test
// in the original terminal
mosquitto_pub -h localhost -t feeds/test -m "Hello World"
// create a password file - replace bob with your username
// the -c overwrites existing file, leave off to append
sudo mosquitto_passwd -c /etc/mosquitto/passwd bob
```

## Create and edit file: /etc/mosquitto/conf.d/default.conf

```
// File: /etc/mosquitto/conf.d/default.conf
allow_anonymous false
password_file /etc/mosquitto/passwd
listener 1883
```

## Test the username and password

```
// restart mosquitto
sudo systemctl restart mosquitto
// restart both sub and pub using username and password above
mosquitto_sub -h localhost -t feeds/test -u "bob" -P "password"
mosquitto_pub -h localhost -t feeds/test -m "Hello World" -u "bob" -P "password"
```

# Node-RED (Detailed: Linux)

```
// Install nodejs and npm
sudo apt install nodejs npm
node -v // validate it was installed correctly

// Install Node-RED
sudo npm install -g --unsafe-perm node-red

// Start node-red service on reboot using PM2
sudo npm install -g pm2

// find and use the correct node-red path
which node-red
pm2 start /usr/local/bin/node-red -- -v
pm2 save
pm2 startup // and follow instructions

// some systems might require
pm2 startup systemd
```

# Securing Node-Red (Detailed: Linux)

```
// Install the node-red-admin package
sudo npm install -g --unsafe-perm node-red-admin
// Create a password hash
node-red-admin hash-pw
```

Edit /.node-red/settings.js

```
// File: ~/.node-red/settings.js
// Find and uncommnet the adminAuth section using the hash
  adminAuth: {
    type: "credentials",
    users: [{
      username: "admin",
      password: "$2a$08$zZWtXTja0fB1pzD4sHCMyOCMYz2Z6dNbM6tl8sJogENOMcxWV9DN.",
      permissions: "*"
    },
    {
      username: "bob",
      password: "$2b$08$W7tKOfDNkm6eZUucgFMoauV1qPwf60MnGzumCc/B8Xj/FUjs8SVMq",
      permissions: "*"
    }]
  },
```

Restart node-red to allow all changes to take effect

```
sudo systemctl restart node-red
```

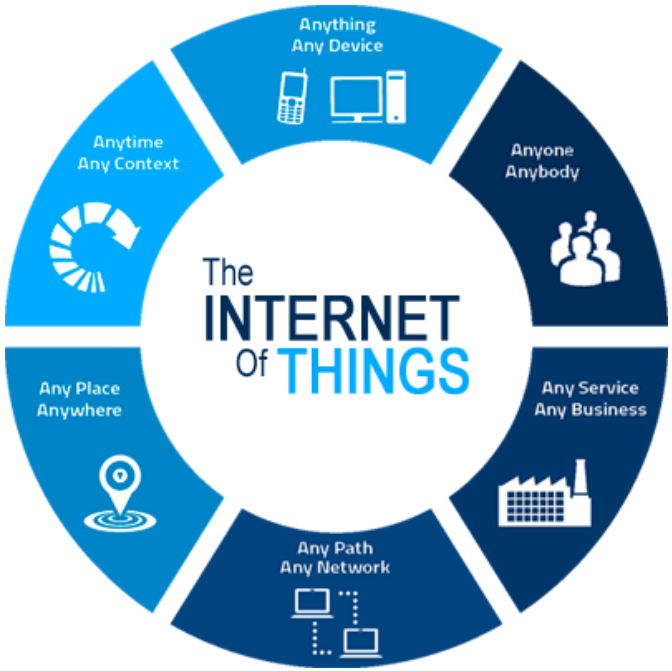
# Accessing from other computers (on same network)

```
// Find your systems IP Address
sudo apt install net-tools
ifconfig

// Open ports from Mosquitto and Node-Red
sudo ufw allow 1880 // Node-Red port
sudo ufw allow 1883 // MQTT port
sudo ufw enable     // Turn on firewall
sudo ufw status
```

- Devices and publish/subscribe to your Mosquitto MQTT Broker if on the same network using your computer's IP address and Port: 1883.
- Access Node-Red from browser on same network using IPAddress:1880

Note: To access anywhere, you need to setup on a cloud service such as DigitalOcean.

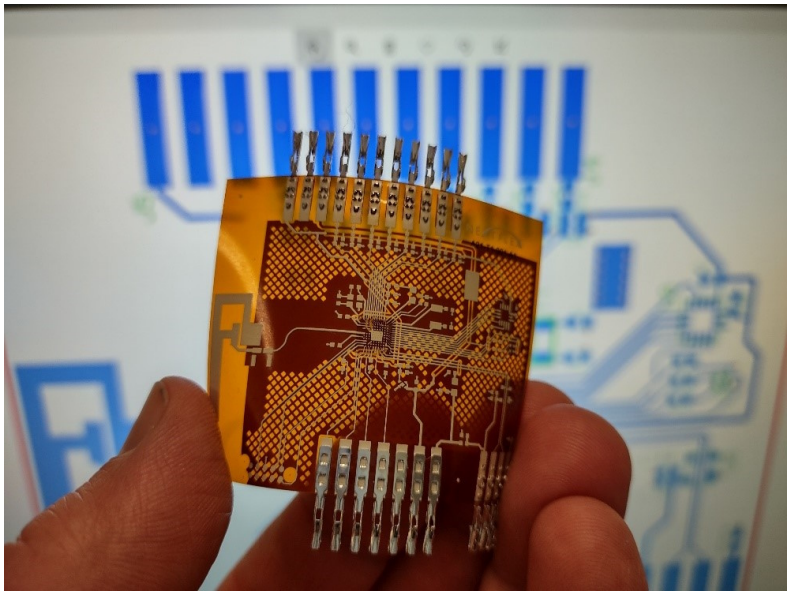
# Module 15 Review




- Learning Objectives
  1. PCB Design
  2. Power and Reset
  3. Watchdogs
  4. Power Management
  5. Cloud Flash
  6. Hosting MQTT/NodeRed

# Module 16: FHE and Debug Mode

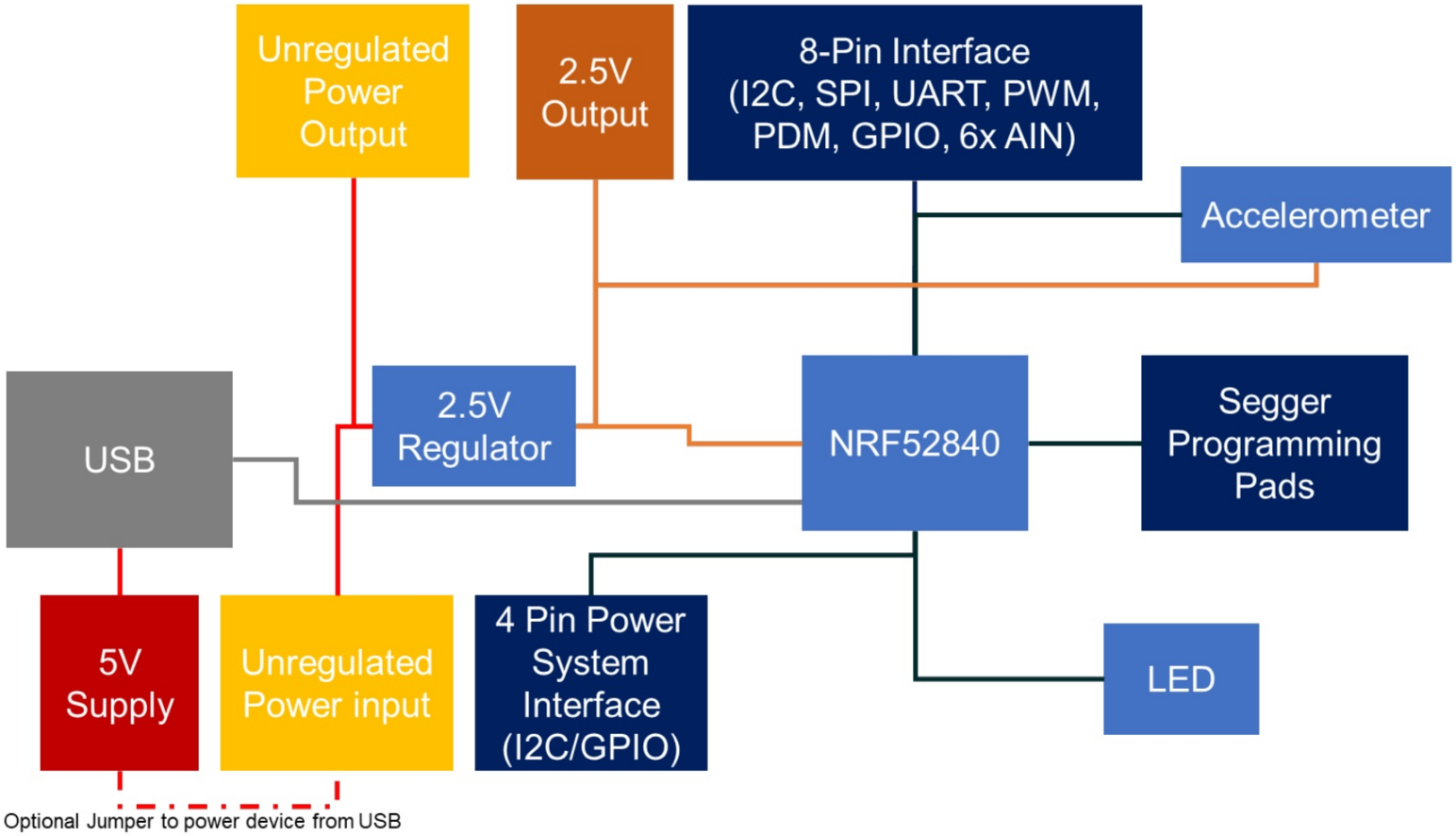
# Flexible Hybrid Electronics (NextFlex A21)



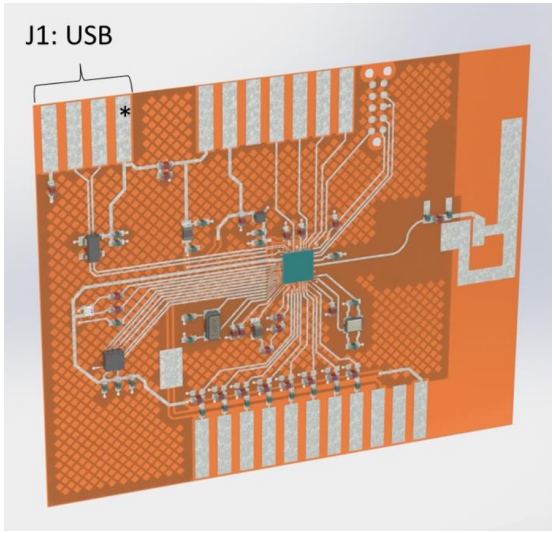
# Why Flexible Hybrid Electronics

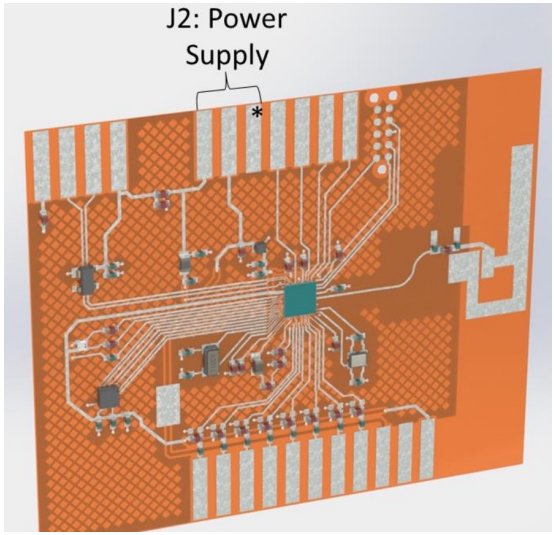
- Structural and/or Conformal Electronics
  - Printing the antenna on the wing of a drone
- Heterogeneous Integration
  - Combining multiple components into one package
  - Multilayer chips
- Real time repair
  - Printing new interconnects or circuit traces
- Environmentally Sustainable
  - Use only the materials you need with minimal chemical processing

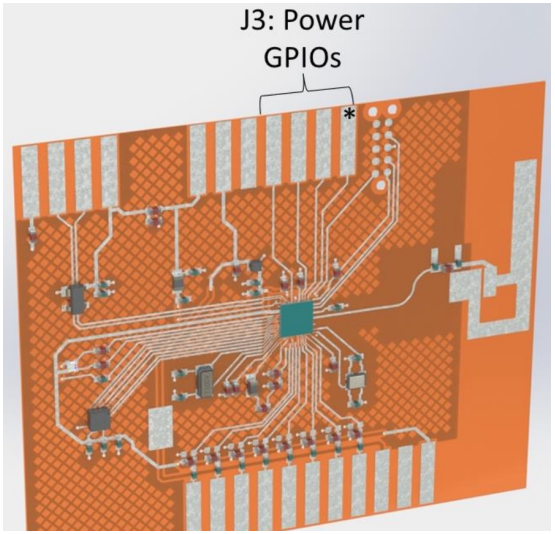
# Flexible Hybrid Electronics (NextFlex A21)

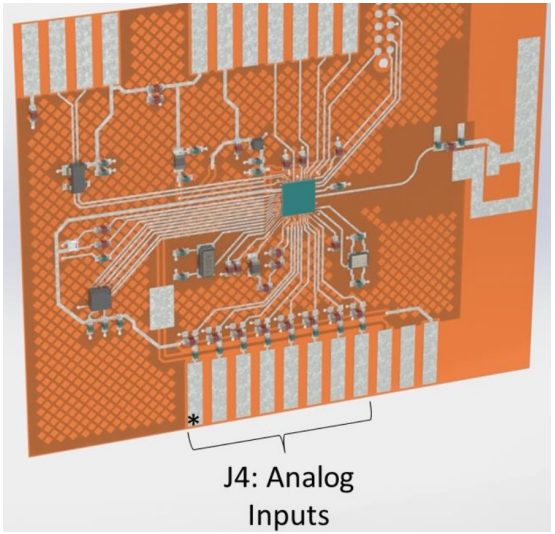


# NextFlex A21 GPIO Pins)

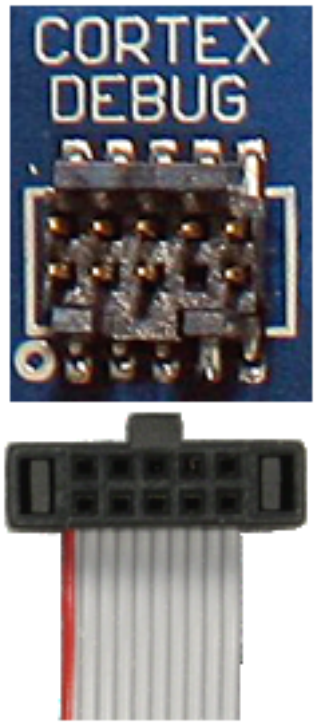


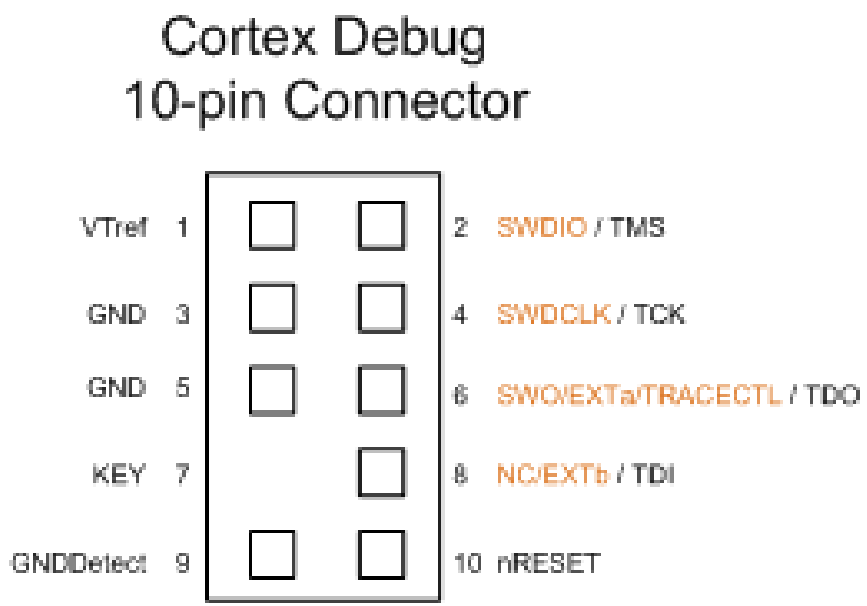



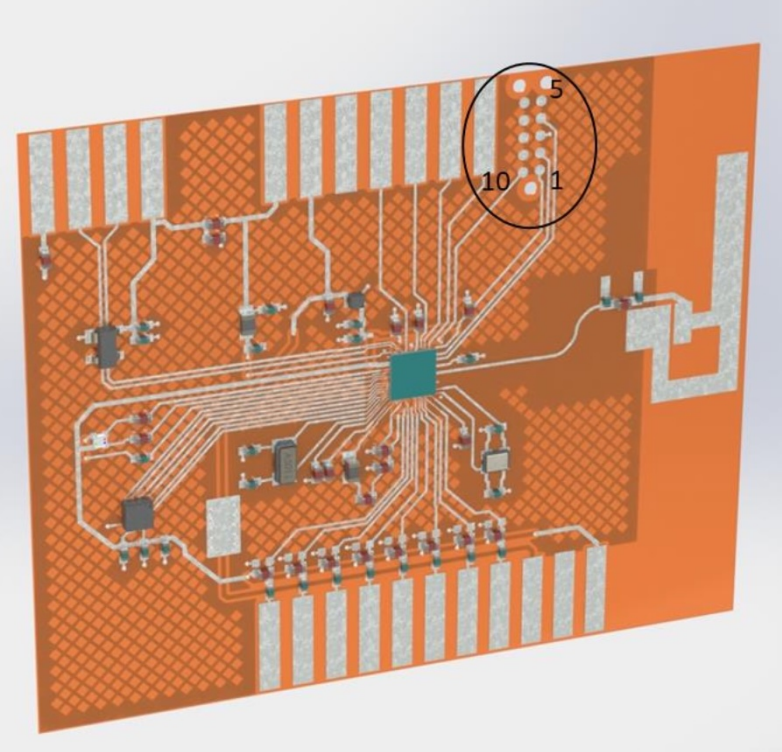



# ARM Cortex 10 pin JTAG Connector

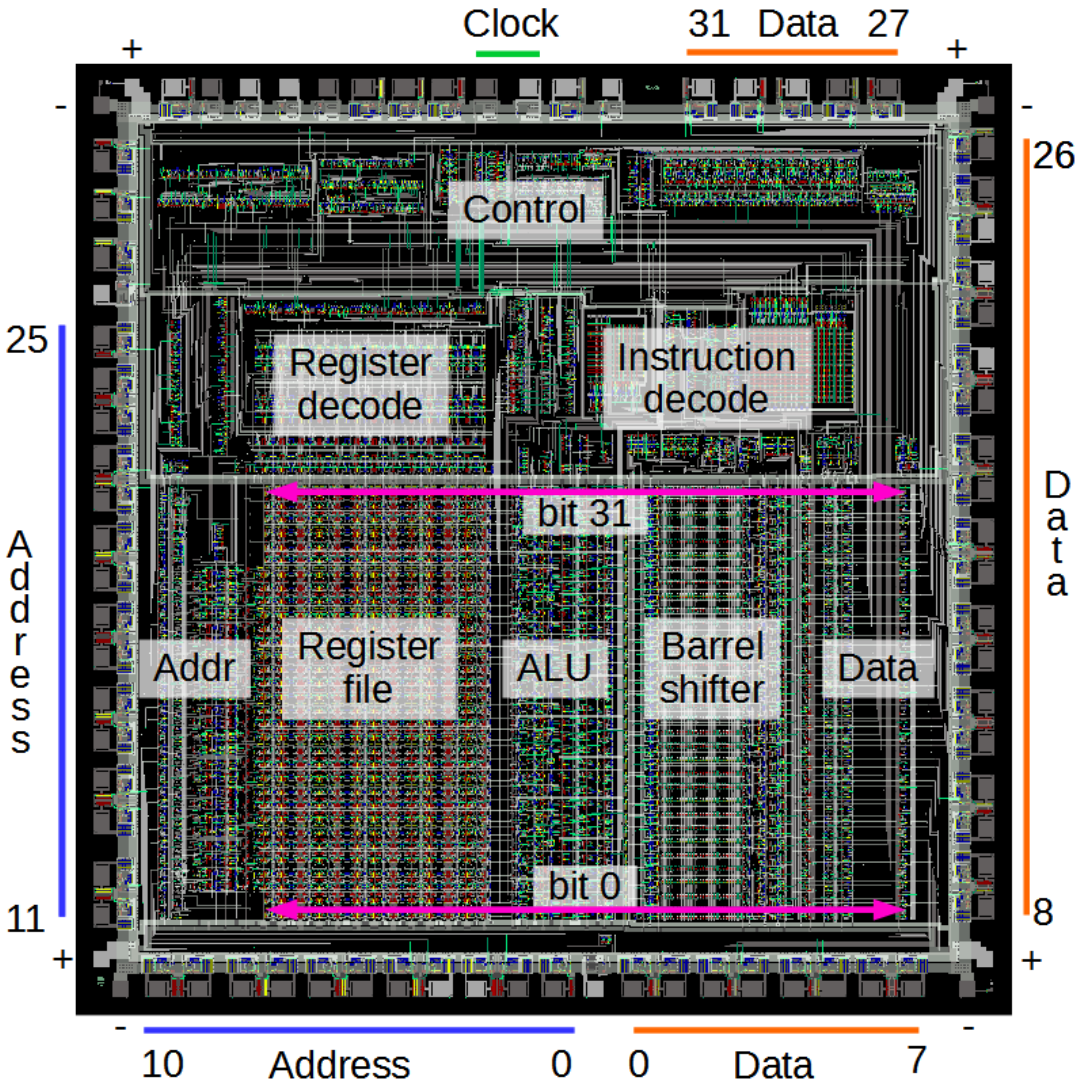


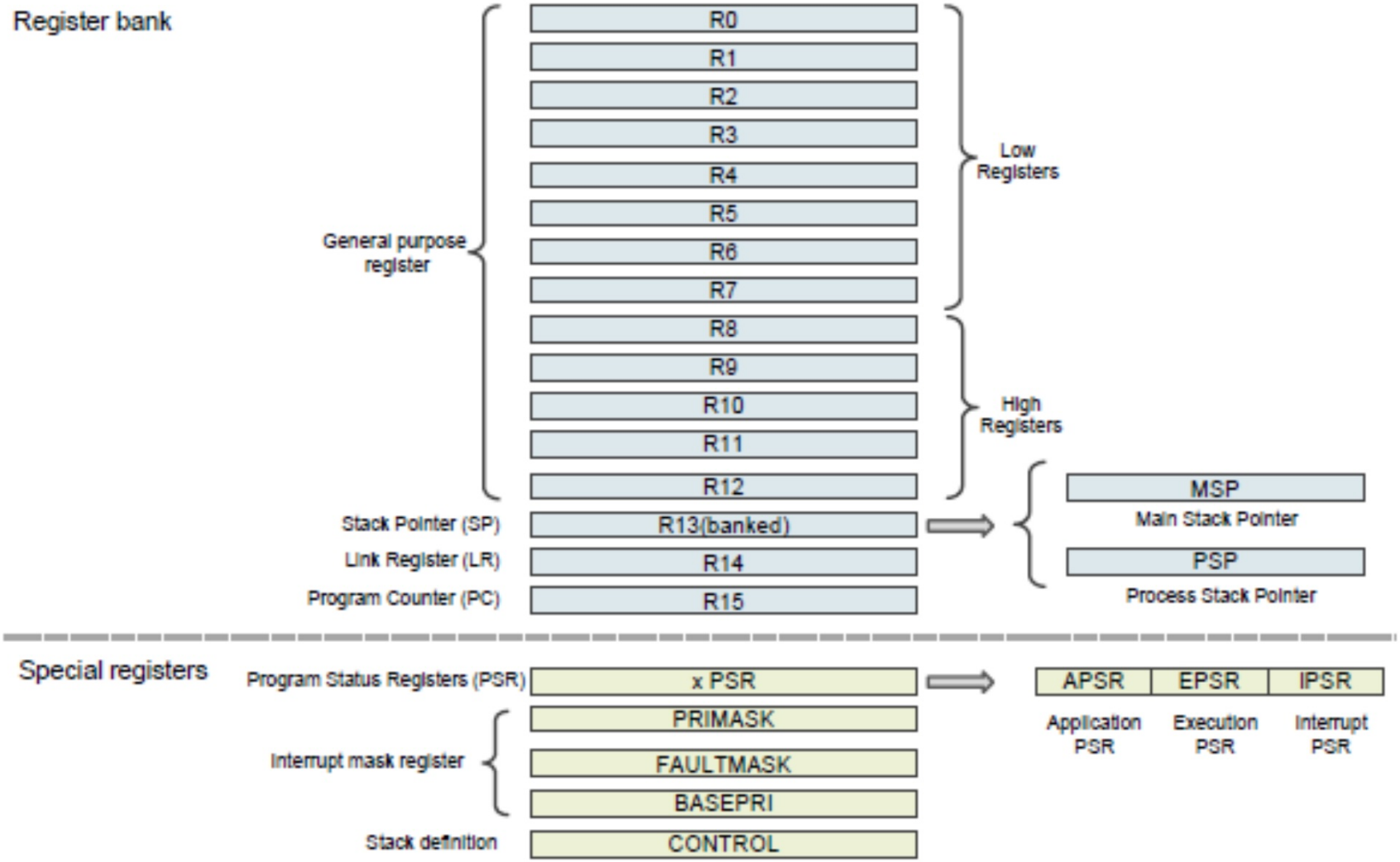
# A21 JTAG programming jig




# ARM Layout (ARM1 for simplicity) - Registers

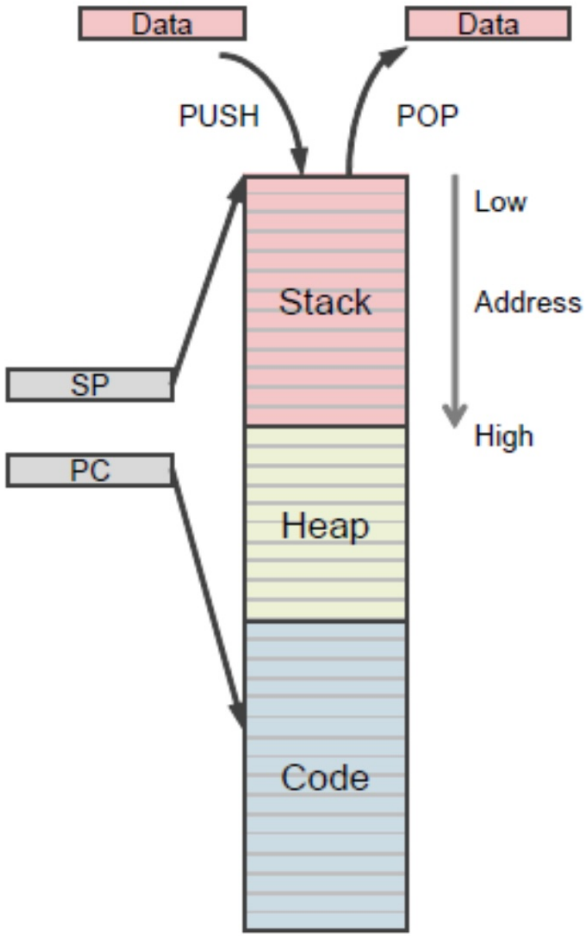


# ARM Cortex M4 Registers



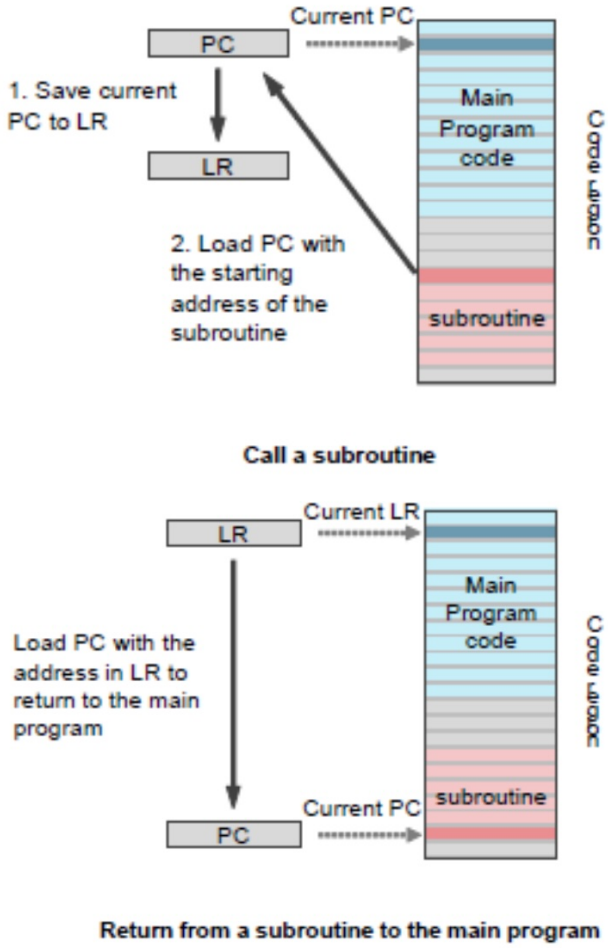
# ARM Cortex M4 Registers

- R0 – R12: general purpose registers
  - Low registers (R0 – R7) can be accessed by any instruction
  - High registers (R8 – R12) sometimes cannot be accessed e.g. by some Thumb (16-bit) instructions
- R13: Stack Pointer (SP)
  - Records the current address of the stack
  - Used for saving the context of a program while switching between tasks
- Cortex-M4 has two SPs: Main SP, used in applications that require privileged access e.g. OS kernel, and exception handlers, and Process SP, used in base-level application code (when not running an exception handler)



# ARM Cortex M4 Registers

- Program Counter (PC)
  - Records the address of the current instruction code
  - Automatically incremented by 4 at each operation (for 32-bit instruction code), except branching operations
  - A branching operation, such as function calls, will change the PC to a specific address, meanwhile it saves the current PC to the Link Register (LR)
- Link Register (LR)
  - The LR is used to store the return address of a subroutine or a function call
  - The program counter (PC) will load the value from LR after a function is finished



# ARM Cortex M4 Registers

- xPSR, combined Program Status Register
  - Provides information about program execution and ALU flags
  - Application PSR (APSR)  item Interrupt PSR (IPSR)
  - Execution PSR (EPSR)
- Application PSR (APSR)
  - N: negative flag — set to one if the result from ALU is negative
  - Z: zero flag — set to one if the result from ALU is zero
  - C: carry flag — set to one if an unsigned overflow occurs
  - V: overflow flag — set to one if a signed overflow occurs
  - Q: sticky saturation flag — set to one if saturation has occurred in saturating arithmetic instructions, or overflow has occurred in certain multiply instructions

# Installing Segger Embedded Studio

1. Install JLink
   - `https://www.segger.com/downloads/jlink/`
   - Check the Box for Use Legacy USB - if that option is available
2. Install Segger Studio
   - `https://www.segger.com/products/development-tools/embedded-studio/technology/installation/`
3. Setup License using License Manager
   - Get MAC address by using: Tools → License Manager → Diagnose problems
   - Request a Free License: `https://license.segger.com/Nordic.cgi`
   - Activate: Tools → License Manager → Activate Segger Embedded Studio

# Assignment: NextFlex A21


1. Open the A21_IoT_Firmware project using File → Open Solution
2. GENTLY place A21 into programming jig, connect to your computer.
3. Target → Connect J-Link.
4. Build → Clean Solution
5. Build → Build Solution
6. Target → Download A21_Demo

Follow along with the instructor to learn how to use Debug mode

# Assignment: NextFlex A21: Modify main.c


1. Create functions to light up the LEDs cyan, magenta, and yellow
2. Modify the variable datatypes from uint8_t to int16_t
3. Increment count by 256 instead of by 1
4. Add printing someNumber and sum to the "count" print statement
5. Create a random(min,max) function
   - Using rand() within the function, allow for a min and max to be set.
   - Modify main(): someNumber = random(0x00BA,0x7000);

# Capstone

# Capstone Projects

Intent:

- Based on a direct observation or need expressed by a guest speakers.
- Original work demonstrating the skills obtained in this class.
- Demonstrate the ability to work as part of a team.
- A pitch to potential employers or investors.

Guidelines:

- Create a Particle (Photon 2 or Argon) based product.
  - ESP32-CAM may be used as secondary controller if camera needed
- Code MUST follow the IoT Style Guide
- Needs to include a Cloud Dashboard component and concepts from Module 15 - In the Wild
- Project will include a video, presentation, GitHub, and Hackster.io.

Previous capstone projects can been found at: `https://www.youtube.com/playlist?list=PL0t2Pk5ETDgxfVptdyr6xbL6MW1-5CJey`

# Extra

# Particle Projects from the CLI

# Create, Compile, Flash from the CLI

```
particle project create
# give your program a name and select N for default project directory
> A new project has been initialized in directory C:\Users\ddcio\Documents\IoT\HelloCLI

cd HelloCLI/src
emacs HelloCLI.ino   #use your favorite editor (notepad.exe, nano, vim, emacs, etc.)

cat HelloCLI.ino
/*
 * Project HelloCLI
 * Description: First Program Creating without VSCode
 * Author: Brian Rashap
 * Date: 09-JUN-2023
 */

void setup() {
  pinMode(D7,OUTPUT);
}

void loop() {
  digitalWrite(D7,HIGH);
  delay(100);
  digitalWrite(D7,LOW);
  delay(100);
}

particle compile argon --target 4.0.2
particle usb dfu
particle flash --usb .\argon_firmware_1686334897288.bin
```

# Particle Libraries

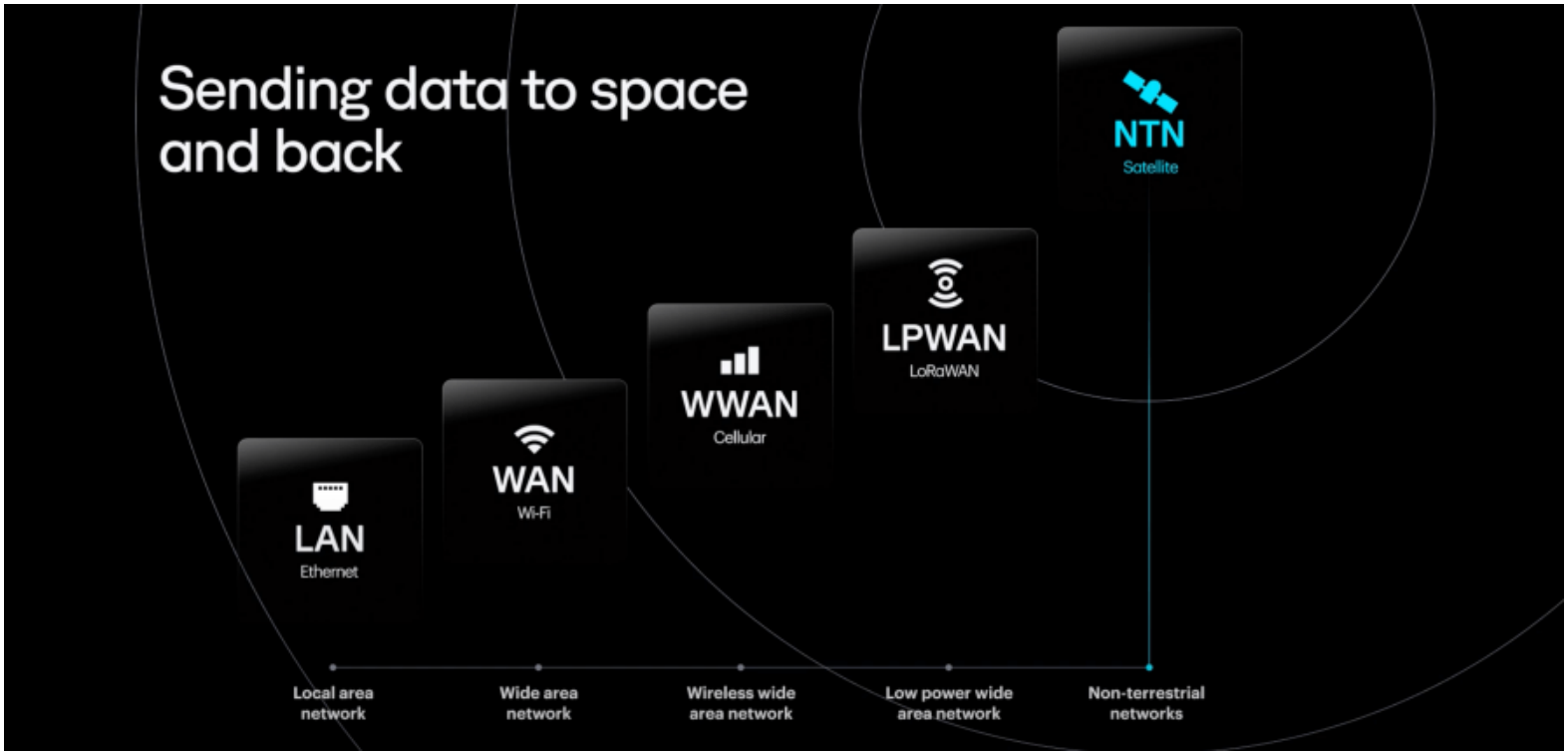
# Creating and Publishing Your Own Libraries

```
// Within your project directory, create a library
mkdir mylib
cd mylib
particle library create

// Modify the Project.Properties file, especially version number

// Create your .h, .cpp, and/or examples within the library directory structure

// Upload your library
particle library upload

// Publish your library
particle library publish mylib

// Note: you can upload/publish new versions
//       just change the version number in Project.Properties
```

# Non Terrestrial Networks (NTN)

# Non Terrestial Networks

Terrestrial networks include cellular networks (2G, 3G, 4G, and 5G), Wi-Fi, LoRA, and low-power wide-area networks (e.g.,LoRaWAN).




Non-terrestrial networks complement traditional terrestrial networks by providing wireless connectivity from airborne or spaceborne platforms, providing the missing link between terrestrial networks.

# Non-Embedded C++

# HelloWorld in C++

```cpp
// include the standar input-output library
// most include statements need the .h, but iostream is an exception
#include <iostream>

// namespace is used to declare regions with the global space
// std enable the standard console (monitor) and input (keyboard)
using namespace std;

char myName[20];

//main is the first function executed in your cpp code
int main()
{
    cout << "Hello World!!!\n";    // cout = output to console
    cout << "What is your name: ";
    cin >> myName;                 // cin = input from console
    cout << "Hello " << myName << ", I hope you are having a nice day.\n";

    return 0;  // denotes successfully executed
}
```

Instructions for installing and writing C++ code in VSCode:
https://code.visualstudio.com/docs/languages/cpp

# The Big Question: What about main()

```
#include <Arduino.h>

extern "C" int main(void)
{
#ifdef USING_MAKEFILE

  // To use Teensy 3.0 without Arduino, simply put your code here.
  // For example:

  pinMode(13, OUTPUT);
  while (1) {
    digitalWriteFast(13, HIGH);
    delay(500);
    digitalWriteFast(13, LOW);
    delay(500);
  }


#else
  // Arduino's main() function just calls setup() and loop()....
  setup();
  while (1) {
    loop();
    yield();
  }
#endif
}
```

# Hello World - What a microcontroller sees

## HelloWorld.ino

```
1 void setup() {
2   Serial.begin(9600);
3   Serial.printf("Hello World! \n");
4 }
5
6 void loop() {}
```

## Hex Code and Assembly Language

```
1  HelloWorld.bin:     file format binary
2  Disassembly of section .data:
3  00000000 <.data>:
4      101c: bd10          pop {r4, pc}
5      101e: 4402          add r2, r0
6      1020: 4603          mov r3, r0
7      1022: 4293          cmp r3, r2
8      1024: d002          beq.n 0x102c
9      1026: f803 1b01     strb.w  r1, [r3], #1
10     102a: e7fa          b.n 0x1022
11     102c: 4770          bx  lr
12     102e: 0000          movs  r0, r0
13     1030: b538          push  {r3, r4, r5, lr}
```

# Useful BASH commands

# Redirect from stdout (standard output)

The > and >> signs are used for redirecting the output of a program to something other than stdout (standard output, which is the terminal by default).

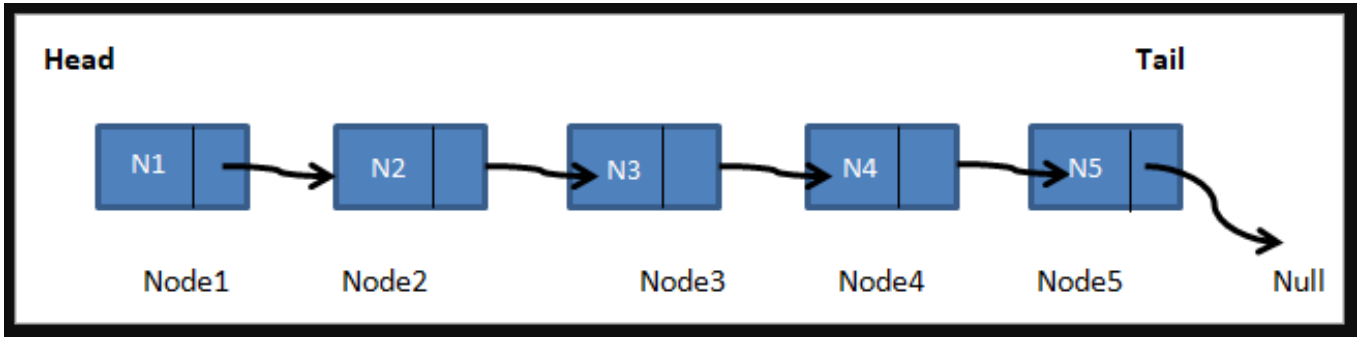
- The >> appends to a file or creates the file if it doesn't exist.
- The > overwrites the file if it exists or creates it if it doesn't exist.

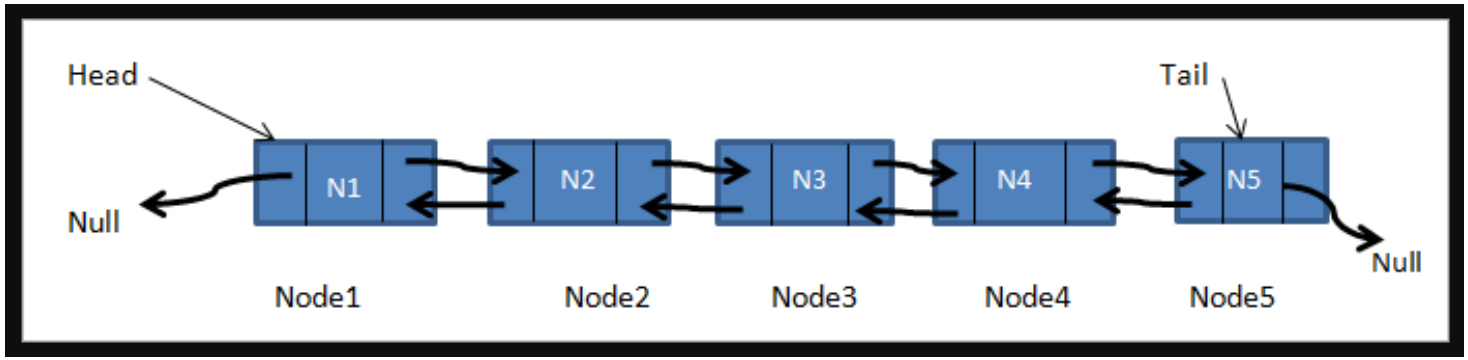
Examples:

```
1
2  # Create a file called "allmyfiles.txt" and fill with the directory listing
3
4  ls > allmyfiles.txt
5
6  # Adds "End the directory listing" to the end of "allmyfiles.txt"
7
8  echo "End of directory listing" >> allmyfiles.txt
9
10 # Create a zero-byte file with the name "newfile.txt"
11
12 > newfile.txt
13
14 # Redirect Particle Serial Monitor output to the file "filename.csv"
15
16 Particle serial monitor --follow >> filename.csv
```

# Linked Lists and Trees

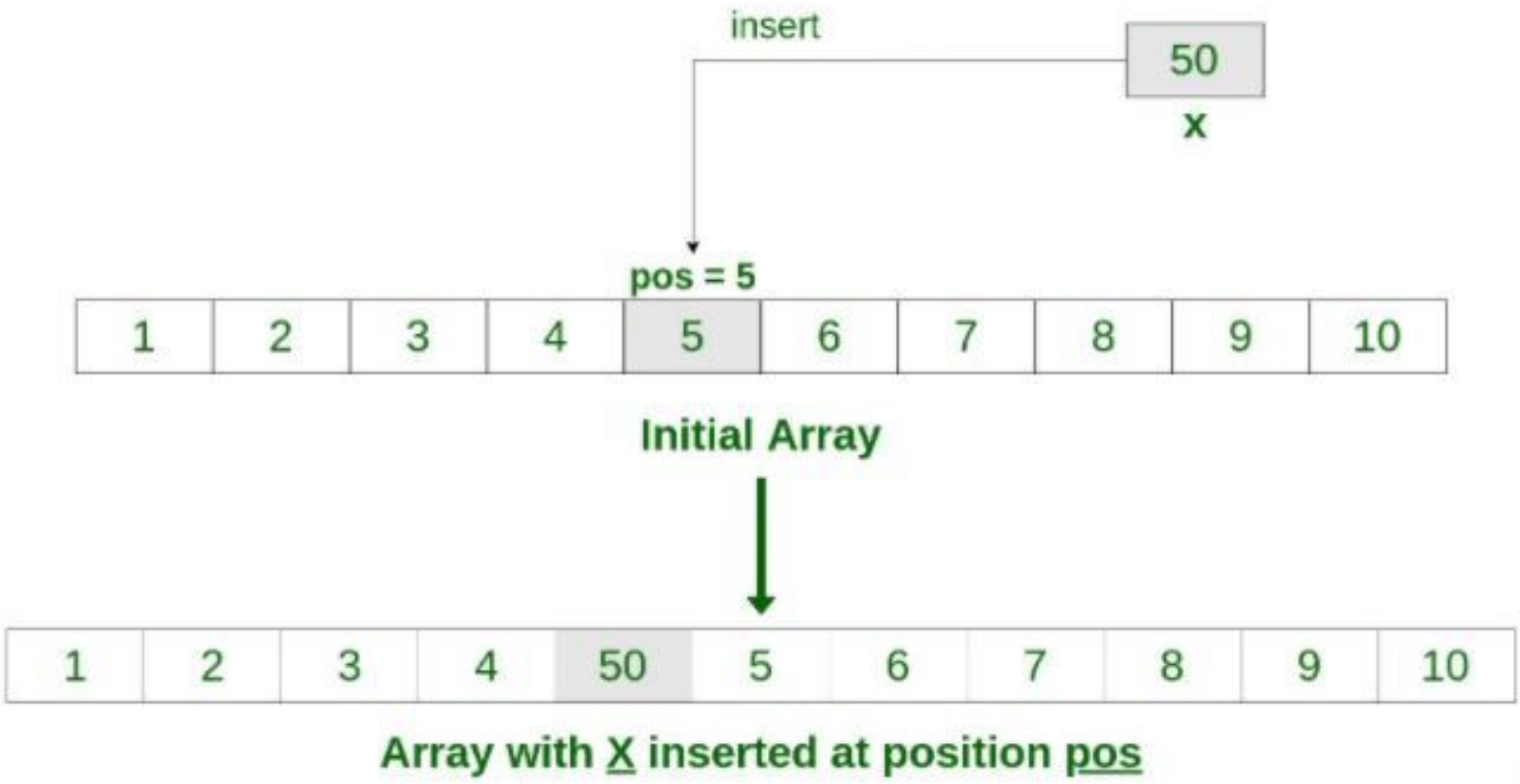
# Linked Lists and Doubly Linked Lists



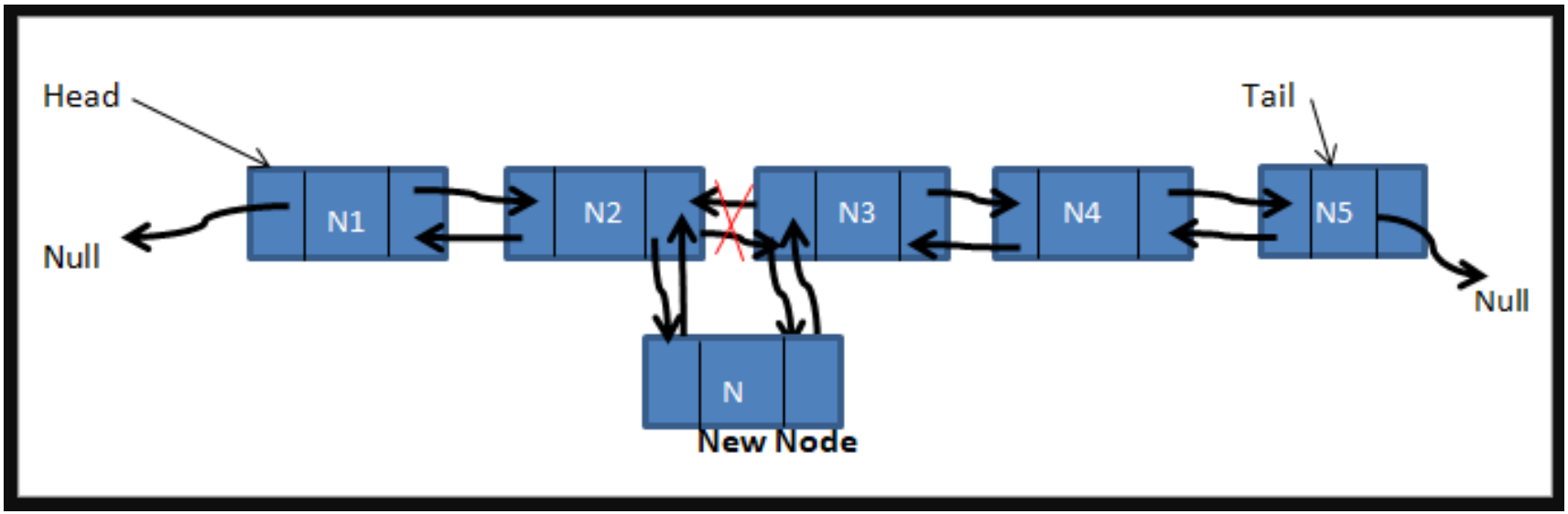




```
1 struct node {
2   struct node *prev;
3   int data;
4   struct node *next;
5 };
```

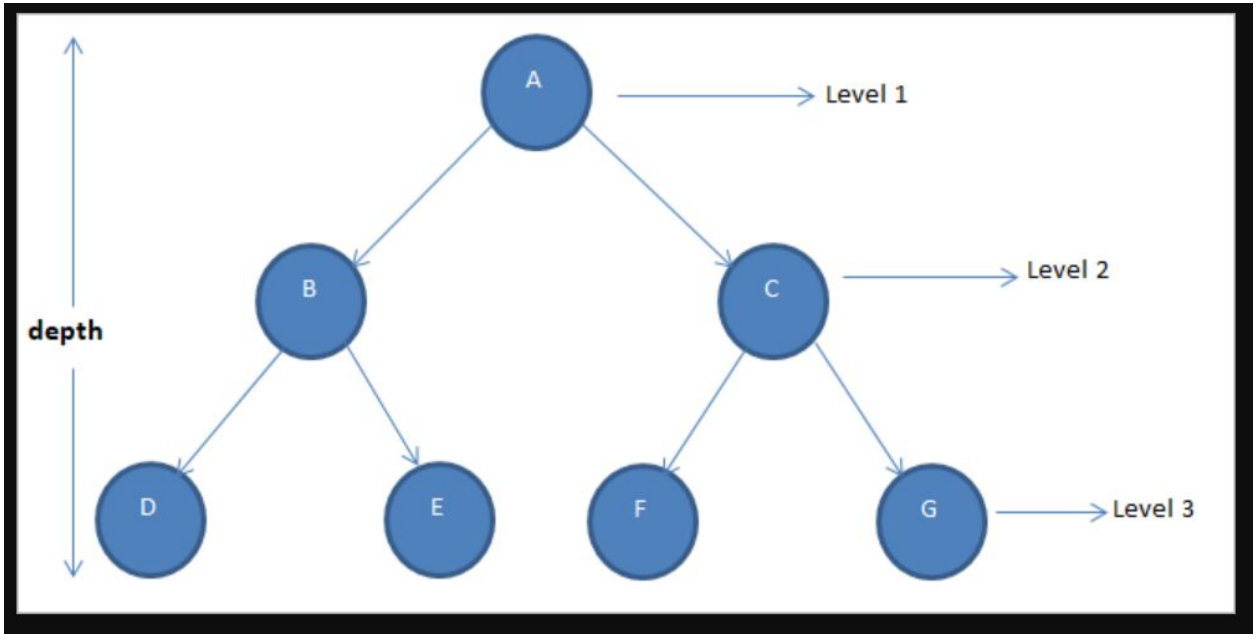
# Inserting a "cell" into an Array



# Inserting a "cell" into a Linked List

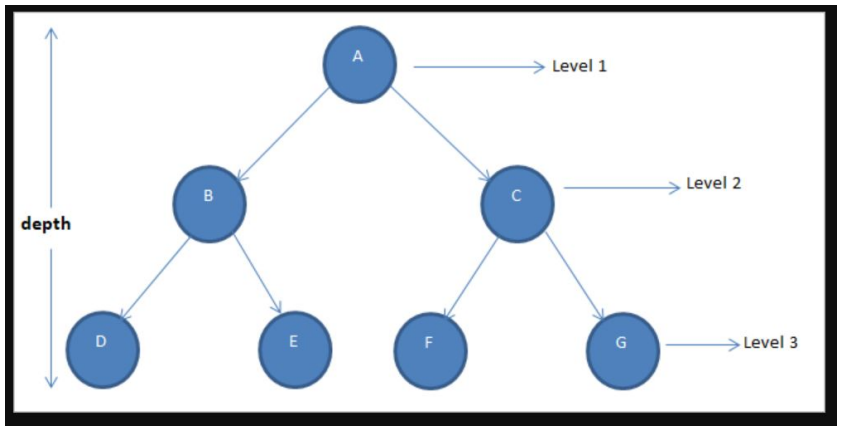


# Binary Trees




```
1 struct bintree_node{
2     bintree_node *left;
3     bintree_node *right;
4     int data;
5 };
```

# Binary Trees




Uses of Binary Trees

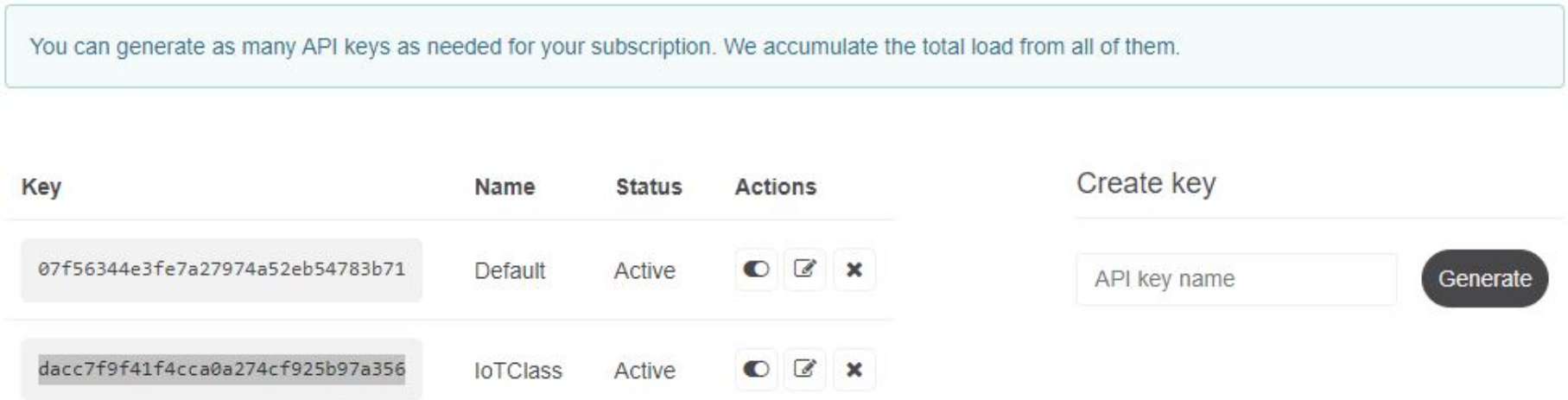
- Binary Search
- Hash Trees
- Heaps
- Huffman Coding
- Syntax Tree

```
1 struct bintree_node{
2     bintree_node *left;
3     bintree_node *right;
4     int data;
5 };
```

# old stuff

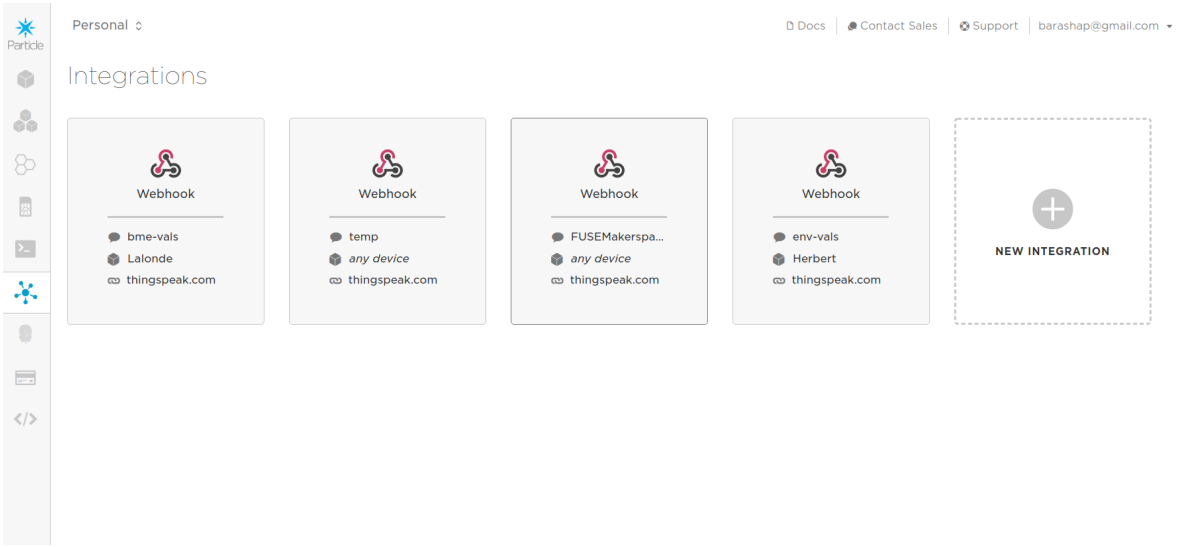
# Step 1 - OpenWeather

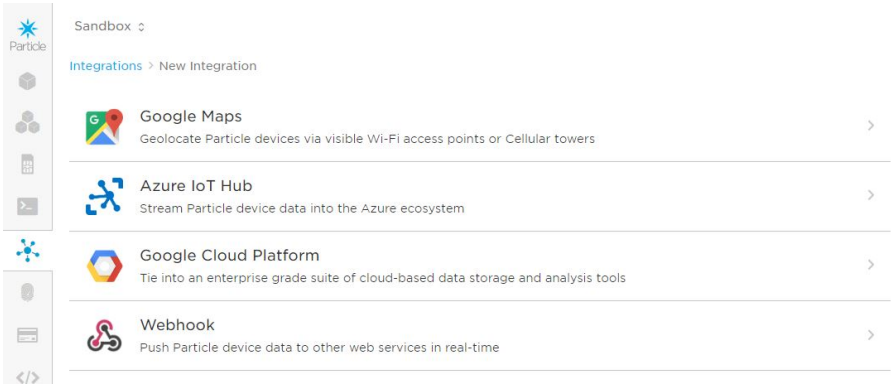
- Create an account at `openweathermap.org`
- Generate an API key at:
  `https://home.openweathermap.org/api_keys`



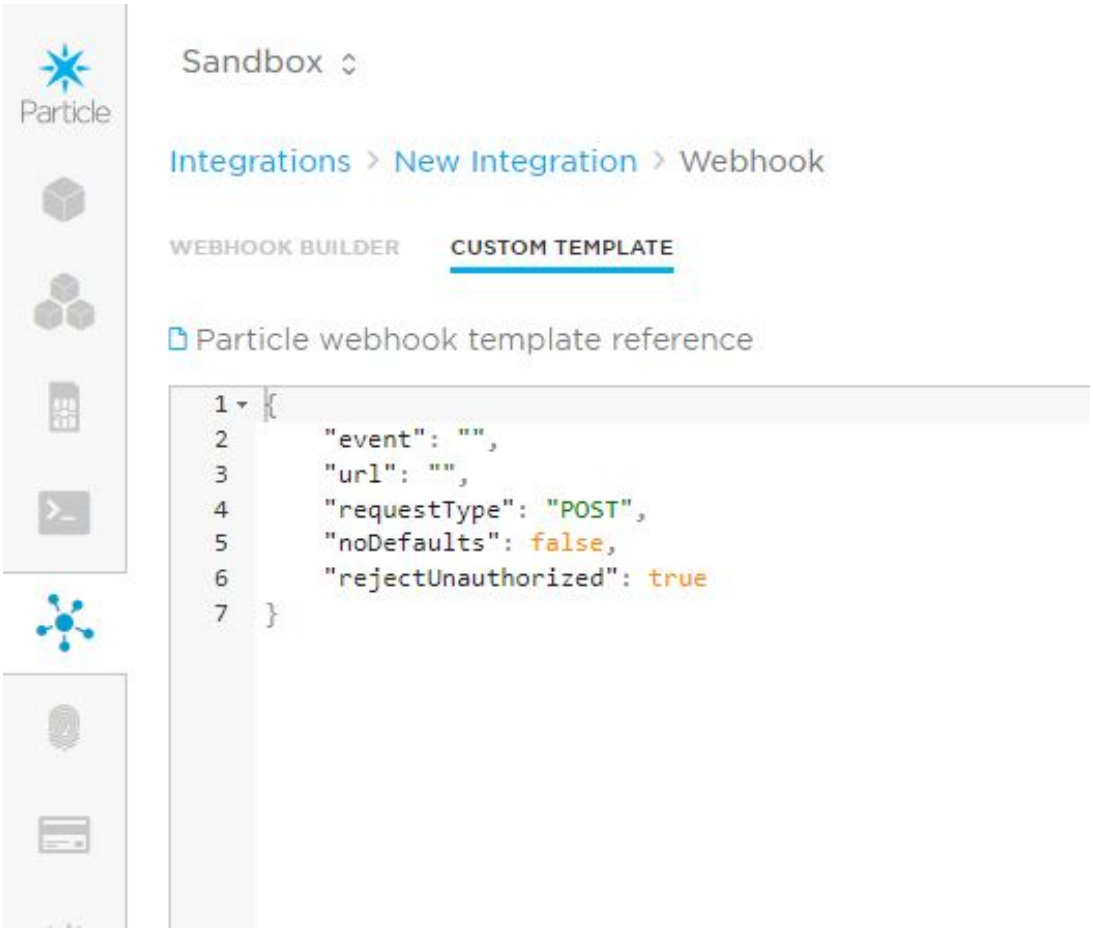
# Step 2 - Create Webhook

From `console.particle.io`:



# Step 3 - Select Custom Template



# Step 4 - Update Custom Template with API format

Adapted from `https://openweathermap.org/api/one-call-api` and `https://openweathermap.org/current`

```
{
    "name": "GetWeatherData for openweathermap.org",
    "event": "GetWeatherData",
    "responseTopic": "{{PARTICLE_DEVICE_ID}}/{{PARTICLE_EVENT_NAME}}",
    "url": "https://api.openweathermap.org/data/2.5/weather",
    "requestType": "GET",
    "noDefaults": true,
    "rejectUnauthorized": true,
    "responseTemplate": "{\"lon\":{{coord.lon}},\"lat\":{{coord.lat}},\"temp\":{{main.
     temp}},\"wind\":{{wind.speed}},\"conditions\":\"{{weather.0.main}}\"}",
    "unchunked": false,
    "data_url_response_event": false,
    "query": {
        "lat": "{{lat42}}",
        "lon": "{{lon42}}",
        "exclude": "minutely,hourly,alerts",
        "units": "metric",
        "appid": "YOUR_APP_ID"  // replace YOUR_APP_ID with your ID
    }
}
```

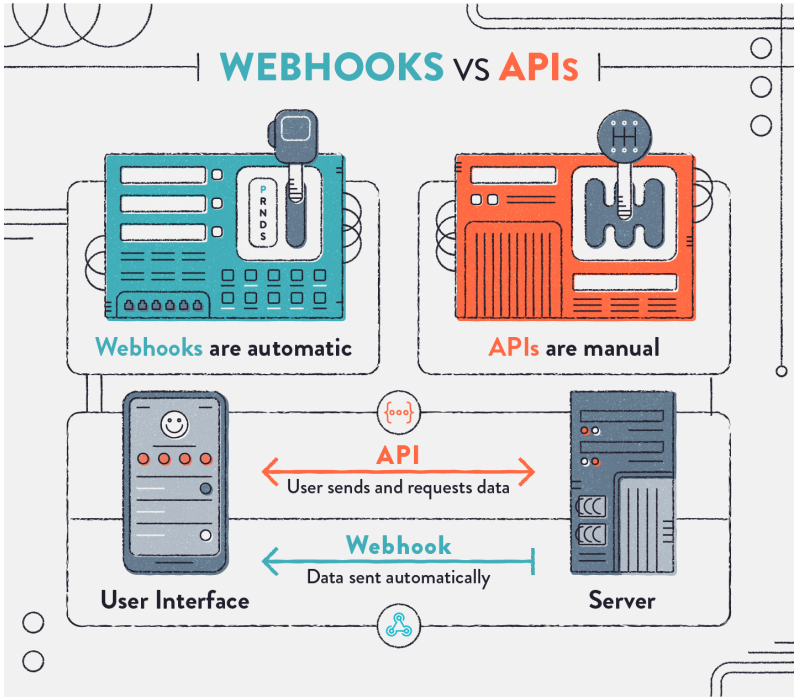
# Step 5 - Particle Code

```
#include "Particle.h"
// Weather Constants and Variables
const char *EVENT_NAME = "GetWeatherData";
float lat42,lon42;
float tempC,wind;
int lastWeather;
String condition;
// Time Variables
int hours, minutes, lasthour, lastminute;

void subscriptionHandler(const char *event, const char *data);

void setup() {
    Serial.begin(9600);
    waitFor(Serial.isConnected,5000);
    delay(500);
    String subscriptionName = String::format("%s/%s/", System.deviceID().c_str(),
     EVENT_NAME);
    Particle.subscribe(subscriptionName, subscriptionHandler, MY_DEVICES);
    Serial.printf("subscribing to %s\n", subscriptionName.c_str());

    Time.beginDST();
    Time.zone(-7);
    Particle.syncTime();
    lastWeather = -9999999;
    lastminute = Time.minute();

    lat42 = 35.08392;
    lon42 = -106.64787;
}
```

# Step 5 - Particle Code

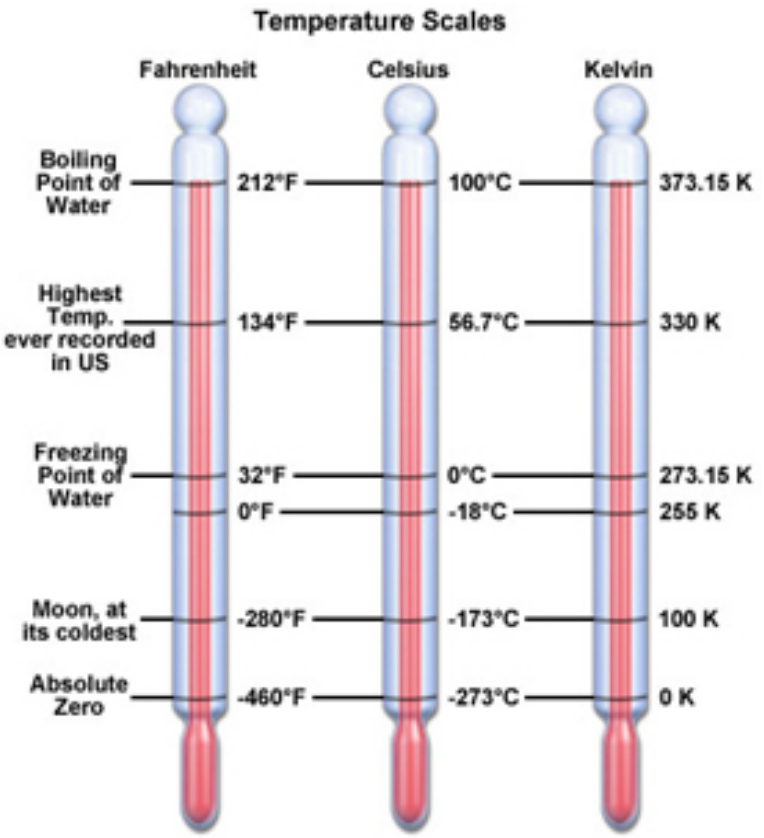
```
void loop() {
    minutes = Time.minute();
    hours = Time.hour();
    if((minutes != lastminute)&&(minutes%10 == 0)) {
        Particle.publish(EVENT_NAME, String::format("{\"lat42\":%0.5f,\"lon42\":%0.5f}",
     lat42, lon42), PRIVATE);
        lastminute=Time.minute();
    }
}
void subscriptionHandler(const char *event, const char *data) {
    JSONValue outerObj = JSONValue::parseCopy(data);
    JSONObjectIterator iter(outerObj);
    Serial.printf("\n\n\nWeather at %2i:%02i\n",Time.hour(),Time.minute());
    while(iter.next()) {
        if (iter.name() == "lat") {
            Serial.printf("Latitude: %0.6f\n", iter.value().toDouble());
        }
        if (iter.name() == "lon") {
            Serial.printf("Longitude: %0.6f\n", iter.value().toDouble());
        }
        if (iter.name() == "temp") {
            tempC = iter.value().toDouble();
            Serial.printf("Temperature: %0.2f(C)\n", iter.value().toDouble());
        }
        if (iter.name() == "wind") {
            Serial.printf("Wind Speed: %0.2f (m/s)\n", iter.value().toDouble());
        }
        if (iter.name() == "conditions") {
            Serial.printf("Conditions: %s\n", (const char *)iter.value().toString());
        }
    }}
```

# Assignment: L09_03_GetWeather




1. Using the OpenWestherMap webhook get the outside weather conditions.
2. Display the OLED:
   - GPS location (hard coded)
   - Indoor conditions from BME280
   - Current outdoor conditions.

# Mapping (or Converting)




Mapping is the conversion from one set of units to another. For example converting from Celsius to Fahrenheit:

$$Temp(^\circ F) = \frac{9}{5} * Temp(^\circ C) + 32$$

C++ provides us with a function to do this mapping:
newVal = map(value, fromLow, fromHigh, toLow, toHigh);

For example:
tempF = map(tempC,0,100,32,212);